SPIRIT WIC Automation Project
DTSD Overview

Covansys Corp.
January 10, 2007

1 Purpose

The purpose of the DTSD is to provide detailed information regarding the tools, technologies, and technical design specifications that are being used to construct the WIC automation system for the SPIRIT consortium.

As eluded to in the RFP this is a "living" document. It is to be kept up-to-date and should document the technical details of the project.

2 Format and Content

The DTSD documentation is comprised of the following artifacts:

- DSTD Cover document (This document)
- Detailed Design Addendum documents
- Software Model
- Data Model
- Disaster Recovery Plan

This document provides a detailed description of the system architecture and provides a road map into the associated Software Model and Data Model documentation provided on CD, each being a component of the overall DTSD. The software model and data model documentation is being provided in HTML format. Using the HTML format allows for streamlined publication and distribution of the information while also allowing the design to hyperlink related information facilitating ease of use.

The DTSD documents are provided in addition to the DFDD. There purpose is to document how the software described in the DFDD is to be constructed. After reading this overview document the reader should be familiar with the architecture being used, the major components and their collaborations and the modeling techniques being applied that provide the detailed information required to construct the software.

After reading a specific chapter of the DFDD, the corresponding detailed design addendum is used to gain an understanding of the software components that related to the interfaces and processes described in the DFDD. The addendum lists the packages and interfaces resources to be constructed and provides a list of sequence diagrams that are used by the developer to understand the runtime object collaborations and responsibilities that must be coded. All of the aforementioned items are located in the Software Model documentation provided on the CD.

Each sequence diagram cites specific objects and messages that are to be exchanged. The semantics of these collaborations are expressed as needed in each class specification and can be easily accessed by double-clicking on the appropriate class in the sequence diagram. Each package in the Software Model also contains one or more communication diagrams that are used to depict the navigation between interfaces.

The Software Model is organized into a number of views. The Use Case View, Dynamic View, Logical View, Component View, Deployment View, and Custom View.

The Use Case View is not being used as of yet since the majority of the functionality that would be expressed as use cases is already documented in the behaviors and processes described in the DFDD. The Dynamic View is used to document business process and data flows. These are provided in the form of Activity diagrams. The Logical View is used to document the data model. The current data model is actually maintained in PowerDesigner and is imported into Enterprise Architect tool for referential reasons. (Note: The team is currently evaluating the possibility of maintaining the data model using Enterprise Architect and discontinuing the use of PowerDesigner.) The Component View contains all of the communication diagrams, sequence diagrams, and class specifications organized into their appropriate packages. The Deployment View documents the software dependencies and deployment strategies. The Custom View is not in use at this time.

The Data Model is provided on the CD. The model contains the ERD and associated table specifications.

Further details in regard to the content and purpose of the Software Model and Data Model can be found in the Software Specifications section of this document.

3 Background

The WIC system is collection of applications that provide functionality that enables a state to provide benefits to participants in the federally funded WIC program. (Women, Infants, and Children) The WIC program provides benefits in the form of food instruments (checks and/or vouchers) and nutrition education to families that qualify to participate in the program. The WIC system is grouped into 4 major functional areas; Participant Management, Vendor Management, Financial Management, and System Administration.

Participant Management
Individuals apply for participation via the applicant prescreening process. The prescreening process assesses the applicant's risk factors and determines if the applicant needs fit within the services being provided. If the applicant does not pass the prescreening process they are placed on a waiting list until such a time that the services provided meet the needs of the applicant. Once the applicant passes the prescreening process a more detailed certification process is performed. Once the applicant is certified they are eligible to participate, (hence the term *Participant*), in the WIC program until their certification period ends. (Basically this means that they can receive program benefits.)

Vendor Management
Participants can receive benefits in the form of food instruments (checks and/or vouchers). Food instruments are non-transferable, non-negotiable documents redeemable for specific food stuffs at participating vendor locations. Locations wishing to participate as WIC Vendors must apply and comply with specific regulations.

Financial Management
The Financial Management portion of the system allows the state to manage the systems general ledger and rebate agreement information.

System Administration
Like any other system, administrative tasks like user maintenance and reference data updates must be performed. In addition, regularly scheduled activities must be conducted in order to

keep the system up-to-date. Typically these activities are conducted at the end of each day, at the end of each month.

End of day activities include, but are not limited to importing and exporting issuance data, bank payment files, vendor survey data, and Food instrument data. This data is distributed to the local state agency, the FDA, the CDC and the appropriate banking institutions. End of month activities generate a number of statistical reports used by the local state agency, the FDA, and the CDC to monitor different aspects of the program.

Ad hoc activities are also conducted on an as needed basis. These activities include, but are not limited to Managing the State-Agency-Clinic hierarchy, Resource availability, Scheduling Group Education and immunization sessions at local clinics, system database management.

Most WIC programs are administered within a state at 3 levels; the State, Agency, and Clinic. A State has one or more Agencies covering mutually exclusive geographic regions within the state. An Agency has one or more Clinics covering smaller mutually exclusive geographic areas with the agency's region. The primary role of the Clinic is to enroll and provide services and benefits to participants. Agencies are focused on managing vendor relationships. The State [office] is primarily concerned with overall program administration.

The SPIRIT project is a consortium of 13 Indian Tribal Organizations (ITOs). These ITOs will operate as 13 independent state agencies but will utilize a single data center. The data center will be hosted by CNI, one of the 13 ITOs. Since each ITO is operating as an independent State Agency the CNI data center will maintain 13 independent logical databases, one for each ITO. Thirteen independent URLs will also be provided, again, one for each ITO, each URL being directed to the ITOs specific database.

4 Project Scope

The scope of this project is to implement a WIC system for the SPIRIT Consortium as agreed upon by SPIRIT and Covansys Corp. The requirements of this system are detailed in number of project artifacts located the project repository. The project scope stipulates not only detailed functional requirements but a number of specific high level technical requirements as well. In addition, Covansys project management has also stipulated a number of technical requirements focused on product reuse for future implementations. As example of some of the requirements are listed below:

- Utilize Smart Client Technology where appropriate to provide feature rich desktop applications that can be updated over the internet.
- Provide the ability for a clinic to "go off-line" for a period of time. Allowing the clinic to check participant information in and out of the central data store. While data is checked out it should treated as "read only" when accessed from the central data store.
- Windows XP Professional will be the desktop operating system.
- Update the overall look and feel of the application where possible leveraging the Windows XP User Interface.
- Use a Service Oriented Architecture in order to provide a set of implementation agonistic services that do not require a low level database connection. These services must be secure and only accessible by users known to the SPIRIT IT domain.
- All software is to be written in VB.NET
- The central data store will be Microsoft SQLServer 2000 Enterprise Edition
- The local data store will be Microsoft SQL Server 2000 Standard Edition

5 Architectural Overview

The SPIRIT project uses a "Rich Client" application that leverages the application independent connectivity of the World Wide Web (typically only leveraged by a thin client application) with the processing power and localization benefits of a traditional thick client desktop application. This combination resolves the concerns of insufficient thin client processing power and excessive "round trips" as well as the client/server data access latency issues traditionally associated with a thick client application. .NET Web Services are used to provide a *Service Oriented Architecture* that communicates with a Central Data Repository while .NET Remoting is used the share information that is exchanged between the various applications running on a client machine. (E.g. user privileges)

Component-Based Development is used to create a set of re-usable software artifacts organized into an n-tiered architecture. The various tiers are described in brief below.

UI Components address the visual needs of the application and provide interfaces to support business process workflows. A rich client application was selected as the best fit for this project given the requirements.

UI Process Components are used to manage the aforementioned user interface workflows. These are somewhat similar to the "controller" in a "model-view-controller" architecture. This particular application implements basic non-modal and modal form management within processes in conjunction with .NET Remoting for inter-process communication and workflow management.

Service Interfaces provide a "store front" that can be used as the focal point for the service oriented architecture. These services provide a layer of abstraction over the business workflows, components, and entities described below.

Business Workflows are used to aggregate a number of business components and interrelated business rules into a logical sequence of events, activities, and/or tasks.

Business Components are used to aggregate a number of business entities into a set of discrete logical transactions each contain pertinent business logic relative to the logic transaction.

Business Entities normally are normally the smallest atomic unit in the solution and typically represent a data attributes aggregated in objects. These objects typically represent a row or collection of rows from a table or view from a data source or an [abstract] object returned from a service invocation.

Data Access Logic Components & Data Services are treated as separate tiers in the architecture, (see diagram below), but are best explained together. The data access logic is typically a set of objects used to exchange data attributes between the client and the central data store. These objects are horizontal in nature and do not have any business context, rather they are set of "generic components" that can be targeted at one or more data sources. Data sources are typically databases or files containing information utilized by the client. The SPIRIT project

Service Agents & Services as well are separate tiers in the architecture but are best described together. Service agents provide the client-side wrapper around consumable services. Services are "published" for consumption either internally, externally, or both. Note the services do not need

service agents in order to be published. This project leverages stateless .NET Web Services that are consumed by the various client applications via service agents. These service agents are capable of routing requests to either the corresponding remote service located at the central data store or to a local data store via the appropriate components from the service interface layer of the architecture. This dynamic runtime routing is based the online or offline state of the applications. In this vein it is evident that the service interfaces published within the context of this application can be considered as external services that can be consumed by other authorized parties.

Communication between the various client applications within a session is achieved using .NET Remoting where appropriate. Communication between the data sources is achieved using a Service Oriented Architecture (SOA). Using an SOA allows the data service to dynamically route the request to the appropriate data source. Service requests routed to remotely located Web Services are expressed as XML and communicated using HTTPS (Secured Hyper Text Transport Protocol). Service requests being routed to local data stores are left in their native business object constructs.

Security with regard to communications is achieved by using SSL to encrypt the XML service payloads being exchanged. Application security is controlled by a specialized application authentication and authorization model that leverages a database driven user profile and role-based privilege model to control access to various application features. Customized credentials are used to exchange tokenized authentication during service invocations.

Software updates are automatically identified at the start of each application session by using a combination of software manifest files and delta database change logic. These updates are then downloaded by the client using practices described in Microsoft's Smart Client Architecture Guide.

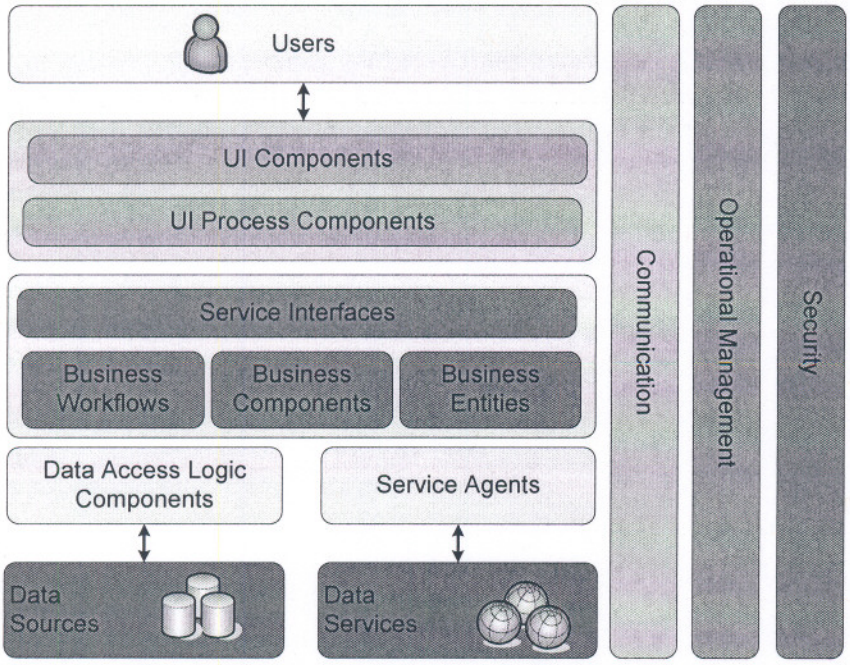


Figure 1 - Microsoft's recommended n-tier architecture

As eluded earlier, this project re-uses the specific components from the tiered architecture on both the Service and Client sides of the solution. This re-use optimizes the development effort and assures that the online and offline behaviors are the consistent.

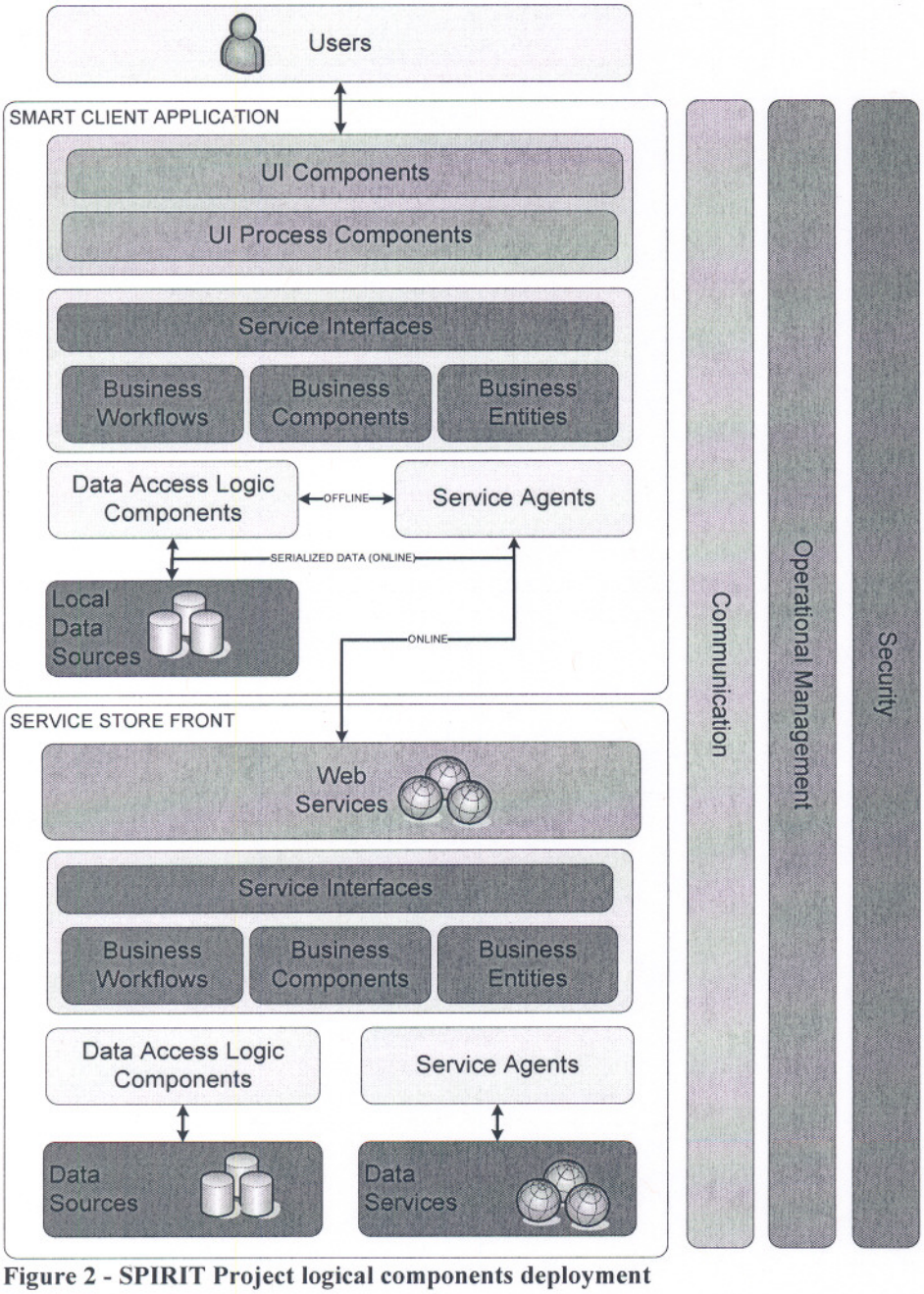


Figure 2 - SPIRIT Project logical components deployment

6 Development Approach

The overall approach is to use the artifacts from the Washington DC WIC project as a basis for the SPIRIT project. This approach dictates a number of steps:

- Publish the DFDD (Detailed Functional Design Document) by using the DC WIC DFDD as a baseline, adding in the new functional requirements stipulated by SPIRIT.
- Build the user interface layouts in order to incorporate images of these interfaces in the DFDD.
- Reuse as much of the existing DCWIC software artifacts as possible. Since the DC WIC software is written using VB6 in a somewhat non-object oriented fashion and interacts with an Oracle database, reuse will be limited and primarily focused on reports.
- Design and implement an n-tier object oriented architecture using Microsoft Web Services as the primary trust boundary between the central data store and consumer (client) applications.
- Optimize the construction effort by providing detailed design packets to construction teams for implementation. Packets will be organized into small focused initiatives allowing for RAD style cyclical development organized in to distinct phases with an emphasis on quality and quick turnaround.
- Utilize the Covansys global network to allow all project team members to access the project artifact repository as needed.
- Use Subversion as the artifact repository.
- Focus development efforts on creating reusable design patterns and frameworks that will provide for a consistent implementation, reduce redundant and repetitive coding tasks, and increase overall quality by creating a bottom up approach for detecting defects.

7 Tools & Technologies

The following tools and technologies are being used to build the system.

Microsoft Visual Studio 2003 is the IDE (Interactive Development Environment) being used to construct the software.

.NET Framework 1.1 Service Pack 2 is the underlying set of windows components being used to interact with the features and functionality of the operating system.

VB.NET is the programming language being used to "write the code".

The **Infragistics NetAdvantage 2004 Volume 2 toolkit** is Third-party commercial product purchased from Infragistics that provides a number of user interface controls being used in the software.

Crystal Reports 8.5 from Crystal Decisions is being used to design and modify reports. The older version is being used in order maintain compatibility with other existing reports since the latest version of Crystal Reports is not backward compatible.

The **CVNS.BPDS.DataAccess Framework** is a software library developed by the Covansys that encapsulates the data access logic needed to interact with the underlying databases. It provides an Object Oriented interface that reduces the need to write dynamic SQL to perform the basic CRUD operations (Create, Read, Update, Delete).

The **CVNS.BPDS Windows Framework** is a software library developed by the Covansys that provides a number of reusable utilities and classes applicable when displaying user interface components

The **Enterprise Architect** modeling tool from Sparx Systems is being used to document the detailed technical design of the software. The tool is being used to provide the Deployment, Sequence, and Class diagrams as well as the Class specifications needed.

Power Designer from Sybase is being used to document the data model. (Note that the team is currently evaluating replacing the use of PowerDesigner with Enterprise Architect.)

CTIMS is an Issue Tracking tool developed by Covansys and is being used to track the design clarifications and defect resolution.

8 Communications

For the purposes of this document, communications can be categorized as the network infrastructure needed to support the system and the software protocols that are being used on the aforementioned network.

8.1 Network Infrastructure

The network being used to support the system is comprised of web servers, database servers, and an application server all being protected by a series of firewalls. As depicted in the diagram below, a router will be used to direct inbound traffic to the IIS Web Server cluster. The cluster will then interact with the database servers by passing through a secondary firewall, essentially creating a DMZ around the web server cluster. The database servers will be configured as an "active/passive" SQL Server cluster, meaning that one server will handle all database requests and the secondary server will provide failover support. In the event of a failover current database transactions will not be completed by the secondary server and must be restarted by the requesting client. The SQL Server cluster will be attached to a SAN. End of Day and End of Month processing will be performed on the EOD/EOM application server. The servers will be managed by dual active directory servers.

The client software will use the Internet to access the IIS Web Server cluster. Each ITO will be responsible for establishing and maintaining connectivity to the Internet for each of there computers, local area networks, and locations.

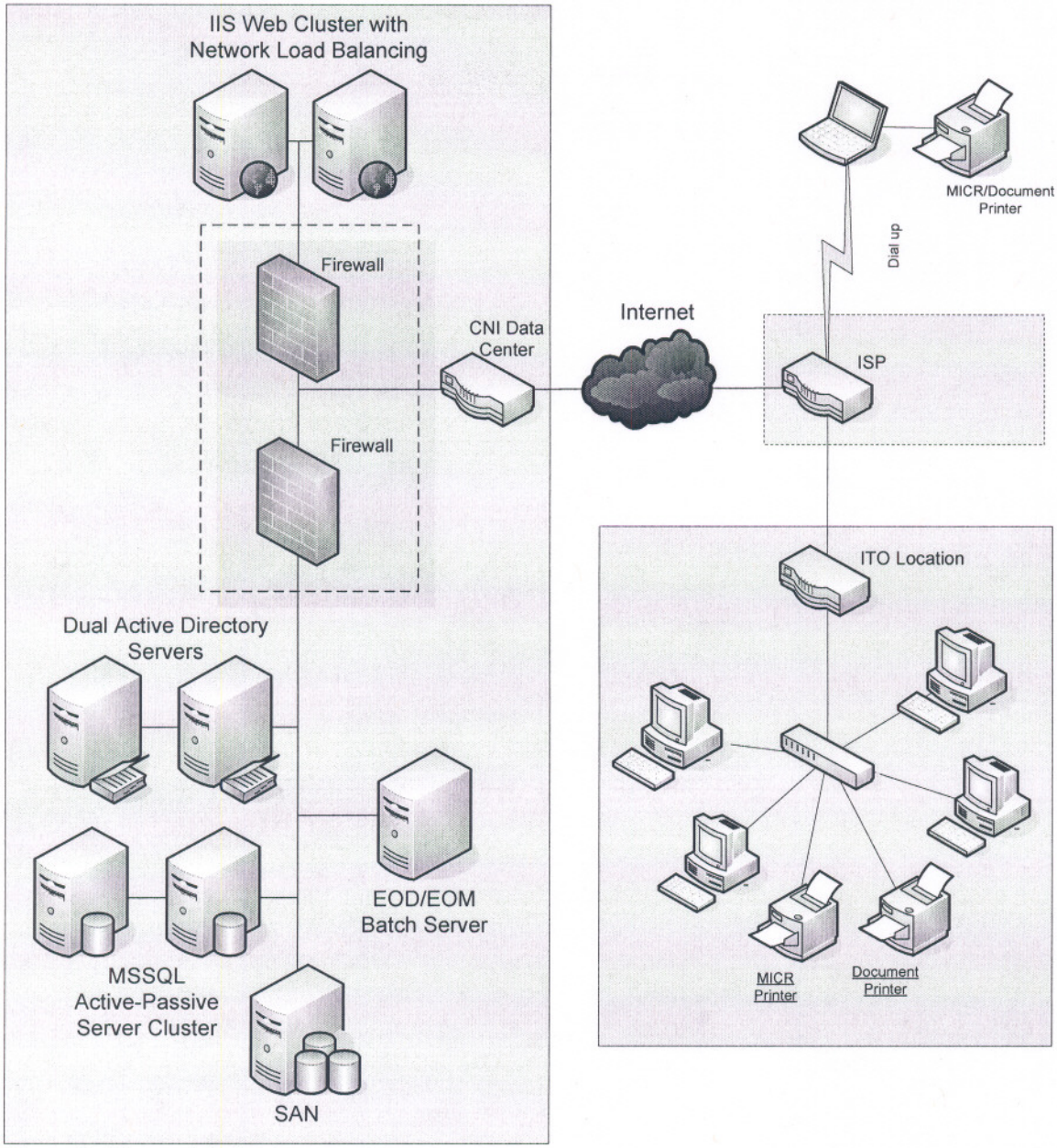


Figure 3 - Network Infrastructure

8.2 Software Protocols

All Communications with the CNI data center with regard to the SPIRIT WIC systems will be performed using the HTTPS (Secured Hyper text transport Protocol) Protocol using SSL (Secure Socket Layer) and certificates purchased from accredited independent third party groups whose certificates will be trusted sources by default within the Internet Explorer web browser. All HTTPS traffic will be directed to the IIS Web Server Cluster.

9 Security

The physical security of the data center falls under the direction and control of the CNI organization. It is recommended by Covansys that only authorized CNI personnel familiar with the hardware and network requirements of the SPIRIT WIC system be granted access the hardware being used to support the SPIRIT WIC system.

Application security will be controlled by the software. Application security, when broken down into its most basic parts, is defined by authentication and authorization. Authentication is making sure someone is who they say they are. Authorization is making sure users can only perform an action for which they are authorized. Between these two parts, application security prevents malicious users from accessing applications or their data at all and friendly users from mistakenly trying to perform actions they shouldn't from within the applications.

9.1 Authentication

Authentication is the process of controlling access to the system. The WIC software uses database authentication to perform this task. Database authentication requires that each individual (user) is included in the user profile within the database. The user profile contains the individual's user id and password. (Note that the password is encrypted for security purposes). There are 2 distinct authentication points within the system. The first is request authentication, the second is application authentication.

Request authentication is performed during each request to the web server. The IIS Web Server is configured to use a customized from of digest authentication (described later in the security section). This configuration mechanism instructs IIS to pass the credentials supplied as part of the http request to a custom application component that in turn uses the information contained in the credentials to lookup the user and verify the information provided against the information in the user profile. If the information matches, the custom application component accepts the credentials and IIS proceeds to process the request. If the credentials are not accepted IIS will reject the request returning an HTTP error indicating the request we refused for security reasons.

Application authentication is performed client-side by the WIC session manager process. During this process the user is prompted to supply their user id and password which is the submitted to the server. Upon successful authentication with the server the WIC session manger will maintain an "open session" on the client. This session contains the profile and privilege information used by the various applications to perform authorization.

9.2 Authorization

Authorization is the process of ensuring the user has sufficient privileges to perform the requested operation. Authorization in this context assumes that authentication has been successfully performed. (See the Authentication section above.)

Role-based security is used to manage privileges. The details regarding setting up and managing roles and privileges are documented in the DFDD. In short, roles have privileges where a privilege is a combination of a feature and an access level. Features are associated to one or more function points in the system. Users are assigned to locations, and at each location are designated as having one or more roles thereby giving the user a set of privileges. Note that this set of privileges is a "super-set" meaning that the highest privilege (access level that has been assigned to each feature) across all the roles assigned to the user at that location is granted.

Authorization is accomplished by each application through the use of a set of permission test calls provided by a UserProfile object maintained in the WIC session process (EXE) made available to each application through the use of .NET Remoting.

9.3 Security Component Architecture

A brief overview of the security component architecture is necessary to effectively use the security component and understand how to transfer existing code snippets to use the component.

9.3.1 Identities and Principals

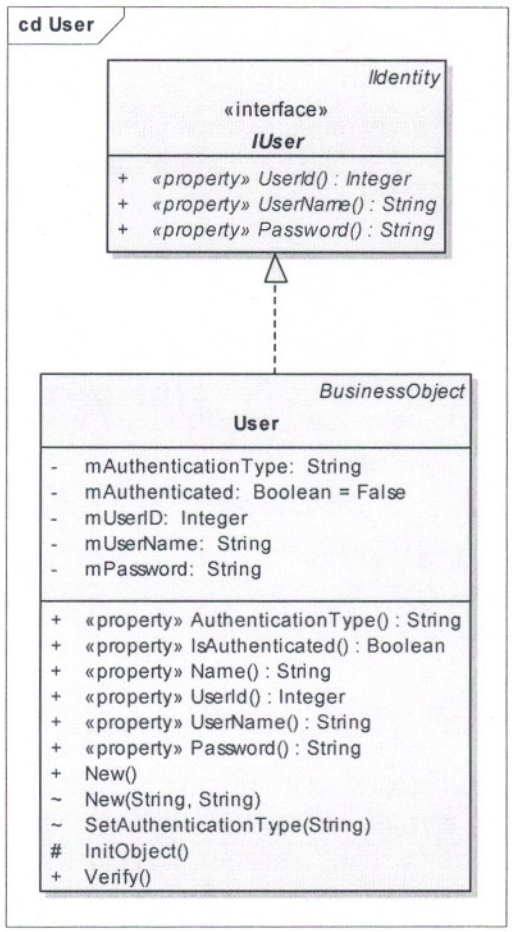
The .NET Framework encapsulates authorization in its Identity and Principal objects and interfaces. The functionality of the objects and interfaces can also be extended to meet more specific needs, as is our case.

9.3.1.1 Identities

Identity objects encapsulate information about a user or entity. The .NET Framework provides an identity interface (IIdentity), a GenericIdentity class, and a WindowsIdentity class. The first two are for building custom identity objects, while the latter is for supporting built-in Windows authentication.

It would seem that we would want to use the GenericIdentity object as a base class to hold information about our users. However, this object is not serializable, a feature we desire for cases where the identity would be remoted. (This functionality is covered in the Single Sign-on section later in the document.) Thus, we are left with creating our own identity class and implementing the IIdentity interface – we extend this with the IUser interface to add functionality – so that our object is treated the same as any other identity.

This is a representation of our identity class:



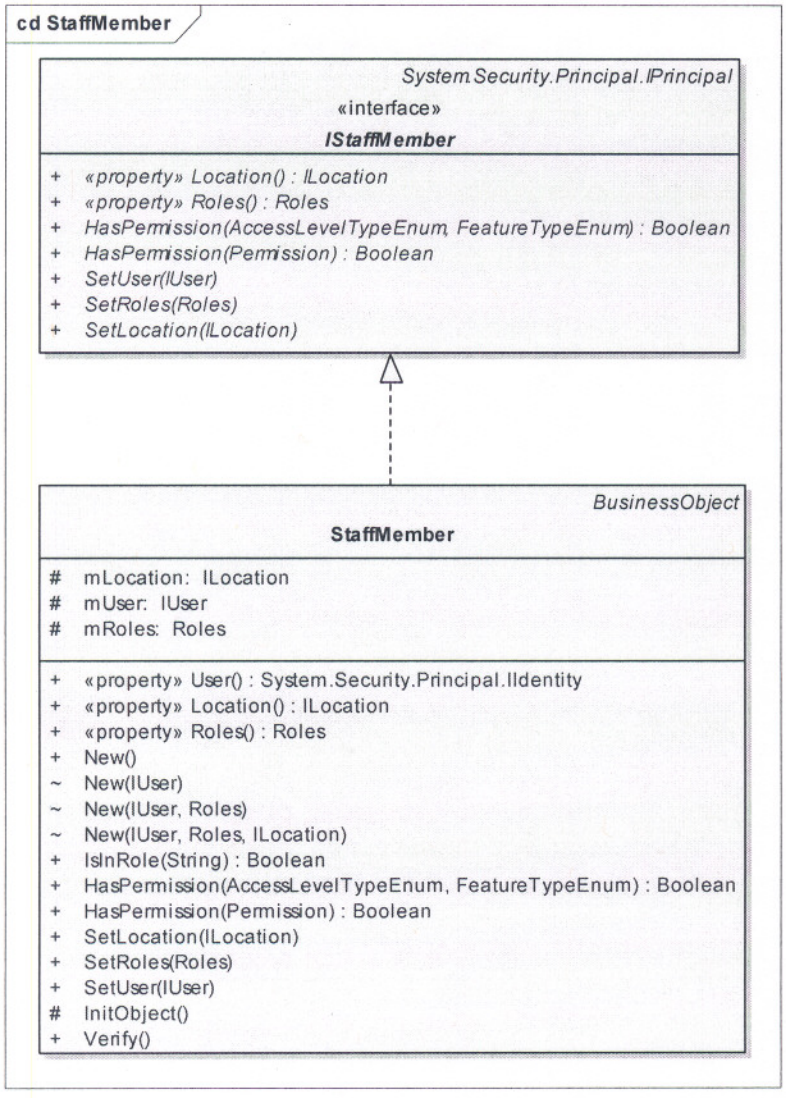
9.3.1.2 Principals

Principal objects wrap Identity objects with further information relating to the user and the context they run the code in. As with Identities, the .NET Framework provides a principal interface (IPrincipal), a GenericPrincipal class, and a WindowsPrincipal class. Again, the first two are for building custom principal objects, while the latter exists to support Windows authentication.

As is the case with Identities, it would seem that we would want to use the GeneralPrincipal object, seeing as how we are not using Windows authentication. However, this object is also not serializable. Therefore, we have created our own principal class that implements the IPrincipal interface.

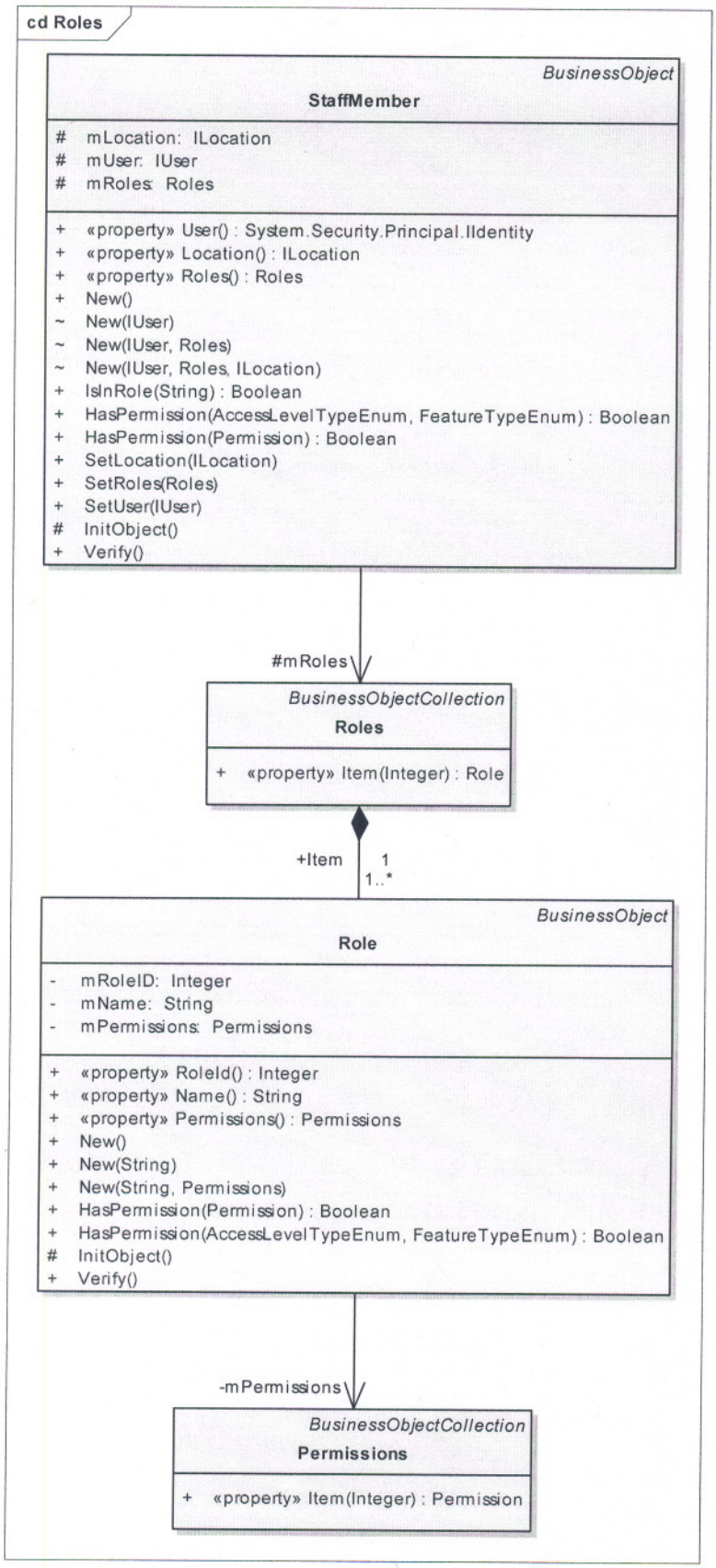
In addition to creating our own principal class that implements IPrincipal, we needed to extend functionality to include permissions checks and considered the Location a User was assigned to. Thus, our principal class implementation, which we'll call a StaffMember, actually implements an IStaffMember interface that extends the IPrincipal interface.

This is a representation of our principal class and its related interfaces:



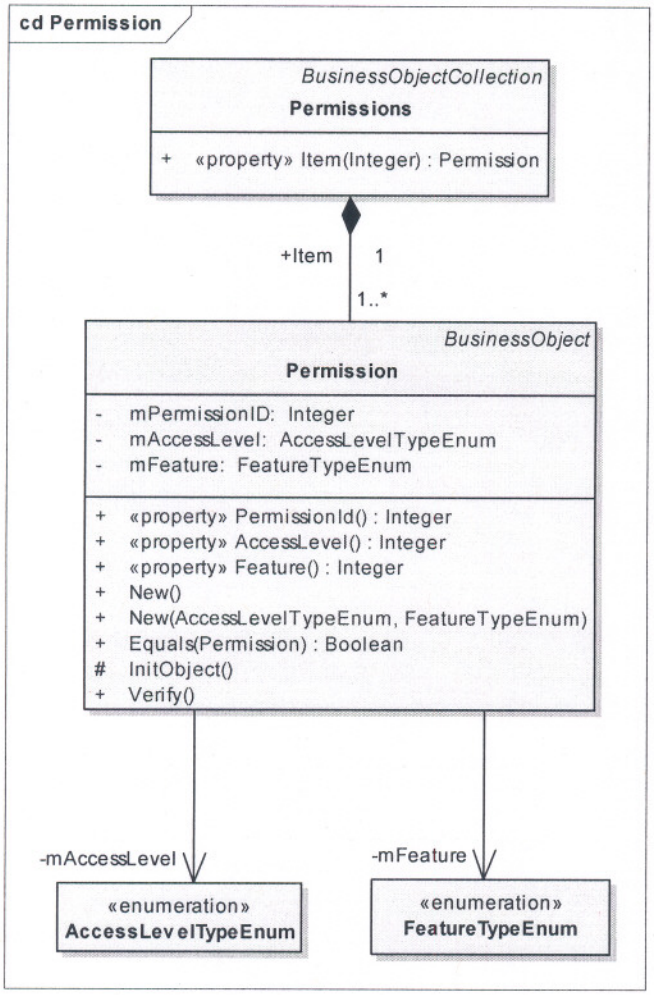
9.3.2 Roles

Roles are simply logical groupings of permissions. Each Principal has a minimum of one role associated with it. A diagram describing these relationships is shown below.


cd Roles
BusinessObject
StaffMember
mLocation: ILocation
mUser: IUser
mRoles: Roles
+ «property» User() : System.Security.Principal.IIdentity
+ «property» Location() : ILocation
+ «property» Roles() : Roles
+ New()
~ New(IUser)
~ New(IUser, Roles)
~ New(IUser, Roles, ILocation)
+ IsInRole(String) : Boolean
+ HasPermission(AccessLevelTypeEnum, FeatureTypeEnum) : Boolean
+ HasPermission(Permission) : Boolean
+ SetLocation(ILocation)
+ SetRoles(Roles)
+ SetUser(IUser)
# InitObject()
+ Verify()
#mRoles
BusinessObjectCollection
Roles
+ «property» Item(Integer) : Role
+Item
1
1..*
BusinessObject
Role
- mRoleID: Integer
- mName: String
- mPermissions: Permissions
+ «property» RoleId() : Integer
+ «property» Name() : String
+ «property» Permissions() : Permissions
+ New()
+ New(String)
+ New(String, Permissions)
+ HasPermission(Permission) : Boolean
+ HasPermission(AccessLevelTypeEnum, FeatureTypeEnum) : Boolean
# InitObject()
+ Verify()
-mPermissions
BusinessObjectCollection
Permissions
+ «property» Item(Integer) : Permission

9.3.3 Permissions

Permissions are a logical representation of a feature to be performed and an access level to that feature for a user. They are represented as follows:



9.3.4 Access Levels

Access levels represent how much access a user has to a feature. They are broken down in an enumeration into the following values:

- **None** – The user has no access to the feature.
- **View** – The user can view the feature.
- **Add** – The user can perform an "Add" action on the feature.
- **Full** – The user can perform all available actions on the feature.

Access levels are inclusive as more access is given to the user. For instance, a Full access level is the same as having a View access level because it includes the View functionality, but a View is not the same as having a Full because Full includes more functionality than just viewing.

9.3.5 Features

Features describe different parts of the application where permissions need to be applied. They are also represented as an enumeration and have the following naming convention: <subject area>_<feature name>. For example, a feature in the Vendor application that allows users to perform actions on vendor applicants might be named Vendor_Applicant.

9.3.6 Single Sign-on

The idea of a single sign-on is that a user's permissions only have to be obtained one time for all running applications. This is a performance optimization to save network bandwidth and latency.

This feature is implemented using a singleton object (Security) using .NET Remoting. Each application makes sure the remoting listener is running before attempting to call the object. Upon the first call to the object, all permissions for that user are obtained and held until the listener application is shut down.

9.4 Configuring Application Security

To assist in understanding the described architecture, this section will walk through the steps of configuring applications to run as described and explain how to implement the security features from within the applications, applying permissions as specified. The Security.Prototype solution has a project called SecurityExample that includes working copies of these examples.

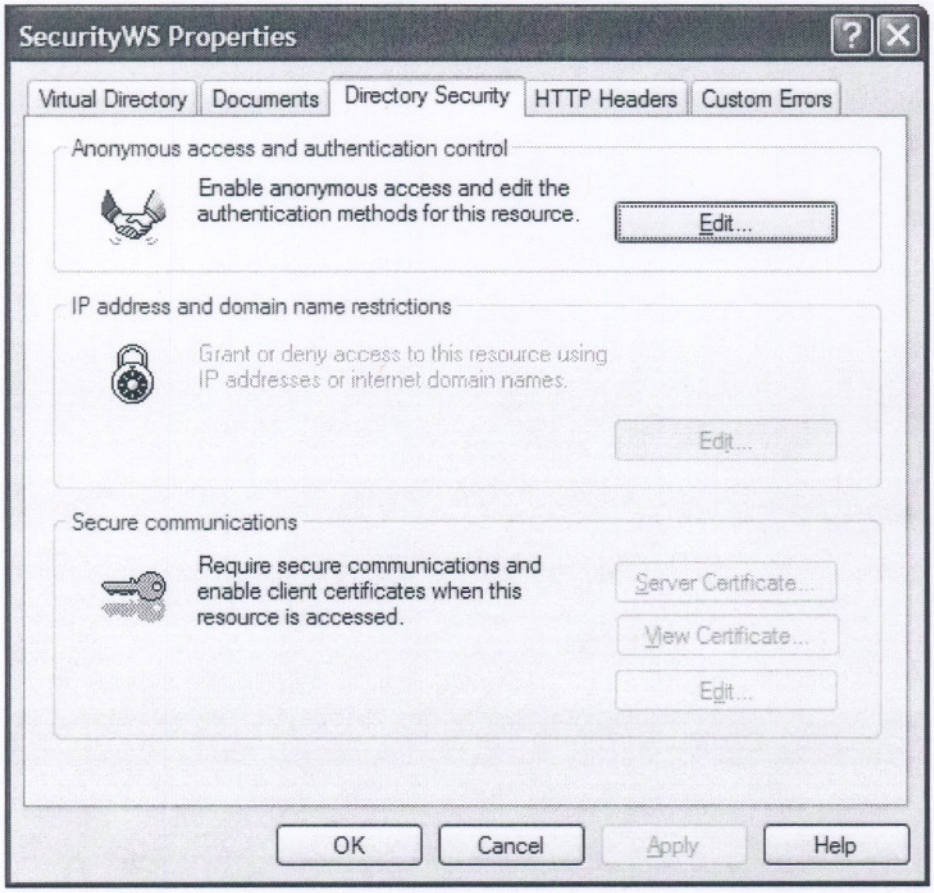
9.4.1 Web Services

Starting from the back-end of the system, ASP.NET web services are hosted in IIS as a web application. Thus, when setting up your web application that hosts web services, the configuration steps that follow should be taken. The ASP.NET web application also requires some configuration settings to be modified in the Web.Config file for the application.

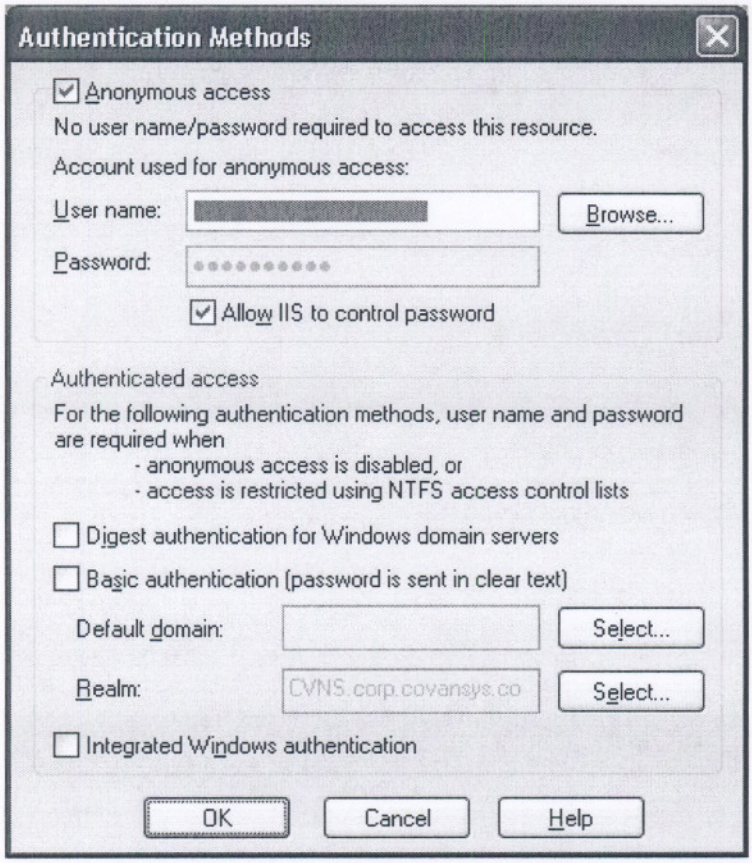
This process is somewhat confusing in that we stated we are using Digest Authentication. In reality, we are simulating Digest Authentication with a custom HttpModule. Thus, the suggested settings look as if we're creating a wide-open system, when we are actually handling things internally.

9.4.1.1 IIS

To configure IIS open the Properties of the Web server and click on the Directory Security tab, as shown below.



Click on the Edit button to open the Authentication Methods dialog.



Make sure only the Anonymous checkbox is check, as shown above and press OK the save all changes.

9.4.1.2 Web.Config

After IIS is configured, configure the ASP.NET web application's web.config file by setting the application's authentication method to Windows.

```
<authentication mode="None" />
```

Then, make sure the application is denying access to unauthenticated (anonymous) users.

```
<deny users="?" />
```

Finally, add the HttpModule that will handle the customized digest authentication.

```
<httpModules>
    <add name="AuthenticationModule"
type="Wic.Services.Security.AuthenticationModule,Wic.Servic
es" />
</httpModules>
```

9.4.2 Client Application

Now that the server is configured properly, the client needs to be set up.

9.4.2.1 App.Config

First, set up the application's app.config file to work with the .NET Remoting listener application. (This listener application is the Security Listener and should be provided to you.) Setting this up is a part of using the single sign-on feature of the system, which looks at the SecurityService.Security object.

The file should look something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.runtime.remoting>
        <application>
            <client>
                <wellknown
                    type="SecurityService.Security, SecurityService"
                    url="tcp://localhost:9999/Security.rem" />
            </client>
        </application>
    </system.runtime.remoting>
</configuration>
```

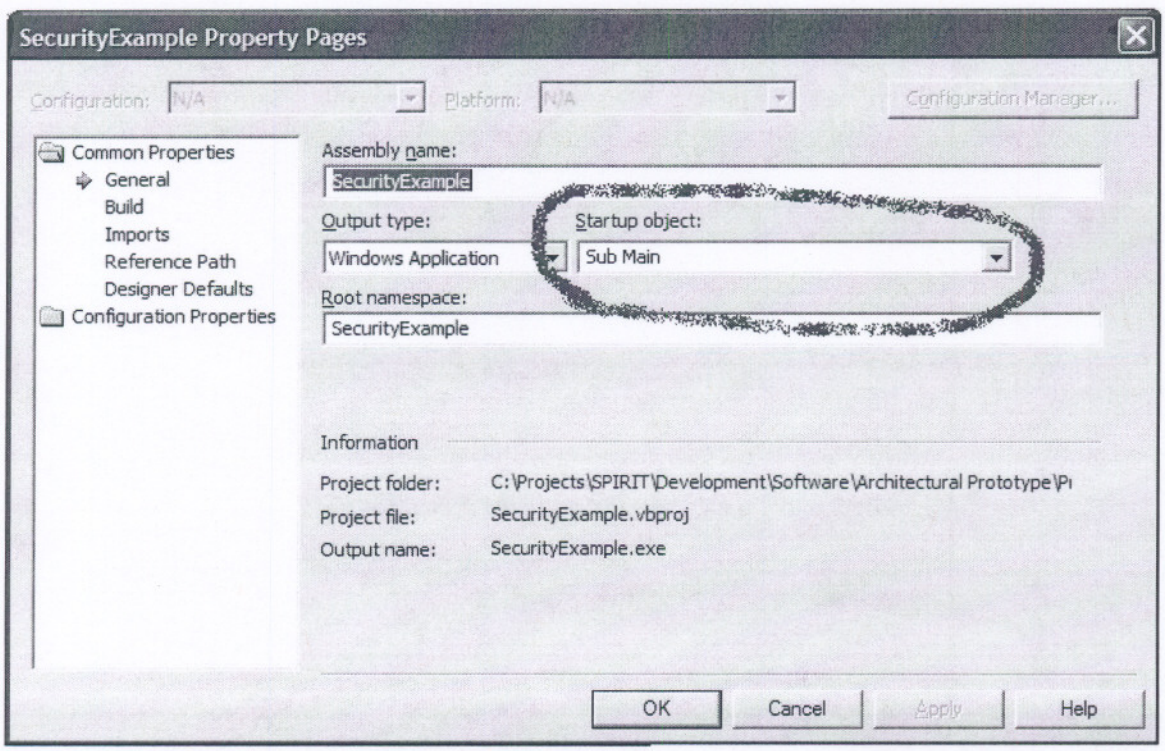
It is likely that the port number and type information are not exactly as shown above, specific information is provided in the appropriate packages and classes in the solution.

9.4.2.2 Startup Class

Next, each application should have a class called Startup with a Main method that looks as follows:

```
Public Shared Sub Main(args() As String)
. . .
End Sub
```

This method then is set in the project properties as the Startup object.



9.4.2.2.1 Configure Remoting

Before the application will "run", the application needs to configured for remoting. (All applications will use remoting for message logging if for no other reason.) Since the remoting configuration settings will be stipulated in the app.config file, the following call must be made to the ConfigureRemoting routine:

```
RemotingConfiguration.Configure( _
AppDomain.CurrentDomain.SetupInformation.ConfigurationFile)
```

9.4.2.2.2 Security Object

A global instance of the Security object is needed in your application so permissions can be checked routinely in the application without having to instantiate a Security object each time. This will be done by creating a shared member variable and read-only property for the Startup class that is initialized either in the Main method or a method that Main calls before running the application. Here is what this might look like:

```
Public Class Startup

    Private Shared mSecurity As Security

    Public Shared ReadOnly Property Security as Security
        Get
            Return mSecurity
        End Get
    End Property

    Public Shared Sub Main(ByVal args() As String)
        RemotingConfiguration.Configure(. . .)
        mSecurity = New Security()
        If (Not mSecurity.CurrentUser Is Nothing)
            Application.Run()
        End If
    End Sub

End Class
```

Since we are using a session manager to manage the login process – see the Special Notes section below for further information – we are assumed to already be logged into the system. But just in case, we must check to make sure the CurrentUser is already initialized.

9.4.2.3 Checking Permissions

Now that everything is set up, when you need to check permissions from within the application, you simply need to have a line or two similar to this:

```
If (Startup.Security.CurrentUser.HasPermission( _
    AccessLevel.Add, Feature.Vendor_ComplianceBuy)) Then
    MessageBox.Show("User has permissions to add a " &
        "Compliance Buy activity")
Else
    MessageBox.Show("User DOES NOT HAVE permissions to " &
        "add a Compliance Buy activity")
End If
```

9.4.2.4 Calling Web Services

You may also need to call a web service to perform an action within the application. Because we are using Digest Authentication on the server hosting the web service, you will need to set the service proxy's credentials so they get pass along with the request. You can do this by calling the SecurityCredentialsFactory method as shown here:

```
Dim Service As New MyService.Service1
Service.Credentials =
SecurityCredentialsFactory.Create("my_user_name",
"password")
MessageBox.Show(Service.HelloWorld)
```

9.4.3 Special Notes

The singleton Security object controlling the single sign-on is not using a caching mechanism to assist with expiring and refreshing its data. Thus, as long as this singleton object stays alive, which it currently has set up as "forever" within the bounds of the session life span, it will keep the original set of permissions. If this caching functionality is required, it should be integrated before release into either a testing or production environment.

As described in the architecture examination, AccessLevels and Features are currently represented as enumerations. These may need to be changed to classes to be more flexible in their implementation. Minor changes in the code would result from this modification, but the core functionality and ideas would act the same.

The Session Manager is a centralized application that will act as a service listener for the security objects that will be shared between applications.

9.4.4 References

In addition to this document, it might be helpful to view the full architectural model for this component. An HTML representation of this is available in the source control repository.

An analysis document also exists that describes why some changes were necessary in moving forward with this security component in light of the security measures taken in the existing DCWIC and other similar applications.

Information regarding securing ASP.NET web services, including detailed descriptions of all authentication and authorization options, can be found at http://msdn.microsoft.com/library/en-us/cpguide/html/cpconsecuringaspnetwebservices.asp.

10 System Components

The software is organized into a number of executables and libraries. Executables are used when a non-modal window must be presented or in independent process must be executed. Examples of these executables, also referred to as processes, are; the ParticipantList.exe, ParticipantFolder.exe, WicSessionManager.exe. Libraries are used to encapsulate functionality use by more than process.

10.1 Session Management

A number of processes and components are used to manage the session and it's associated background processes. The session is essentially the "wrapper" around the WIC applications. Although each application is run in its own process space, each is dependent on interacting with the session process (WicSessionManager) at runtime. In conjunction with the session is the WicAppLoader which is used to launch the session and each of the applications as needed.

10.2 Starting the Applications

The WicAppLoader process encapsulates the functionality necessary to start other applications. As an example of how this application works, a shortcut would be placed on users' desktops to start up the ParticipantList application. The shortcut, however, does not directly start the ParticipantList application. Instead, it points to the WicAppLoader and passes ParticipantList.exe as an argument to the application. The WicAppLoader then makes sure everything is ready for that application to run – makes sure the latest updates have been downloaded, starts the WicSessionManager, makes sure the user is authenticated with the system – before starting the application.

This application is implemented as a Windows Forms application with no user interface. This is done so that it is transparent to the user using the system.

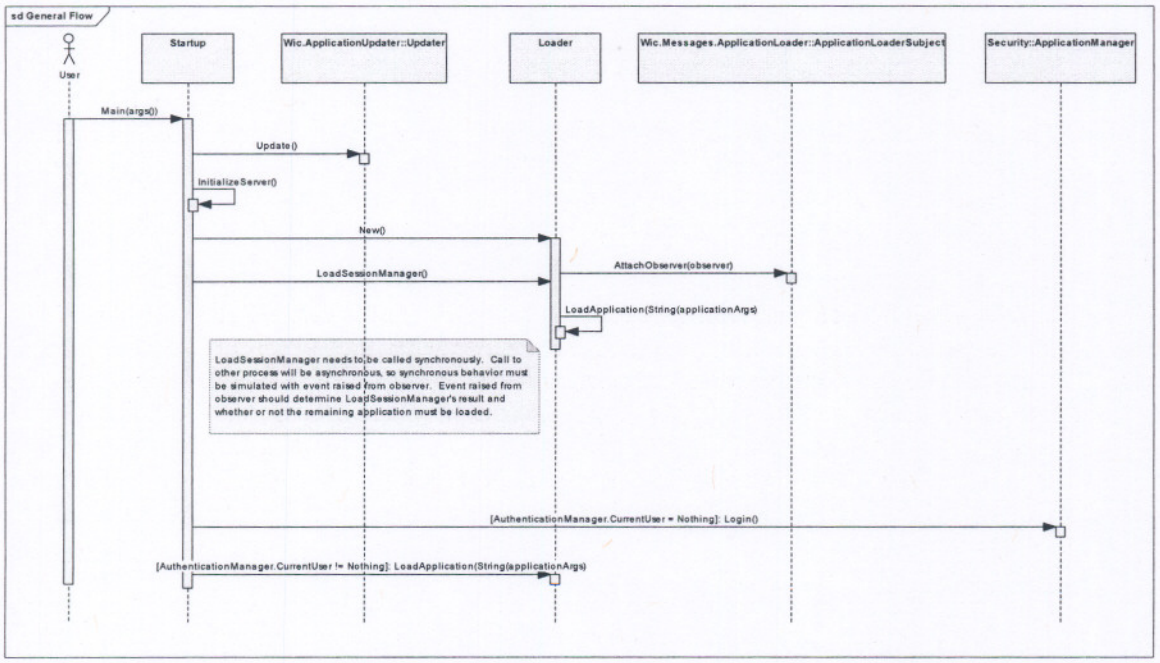


Figure 4 - Starting an Application

The diagram above depicts the general sequence of events and interactions that occur when starting an application. This sequence diagram can be found in the WicAppLoader package in the software model.

10.3 Managing the Session

Only one session (WicSessionManager process) can be running on a client machine at any given time. As stated above, upon invocation of the first application (through the WicAppLoader) the

session is started prompting the user to sign on. Once signed on the session retrieves and maintains the user profile information containing the authentication and authorization information needed by the other applications at runtime. Each application (process) is can then access this information via .NET Remoting as such the WicSessionManager process essentially acts as a host for the remoted objects that are shared between all WIC applications running on a single machine.

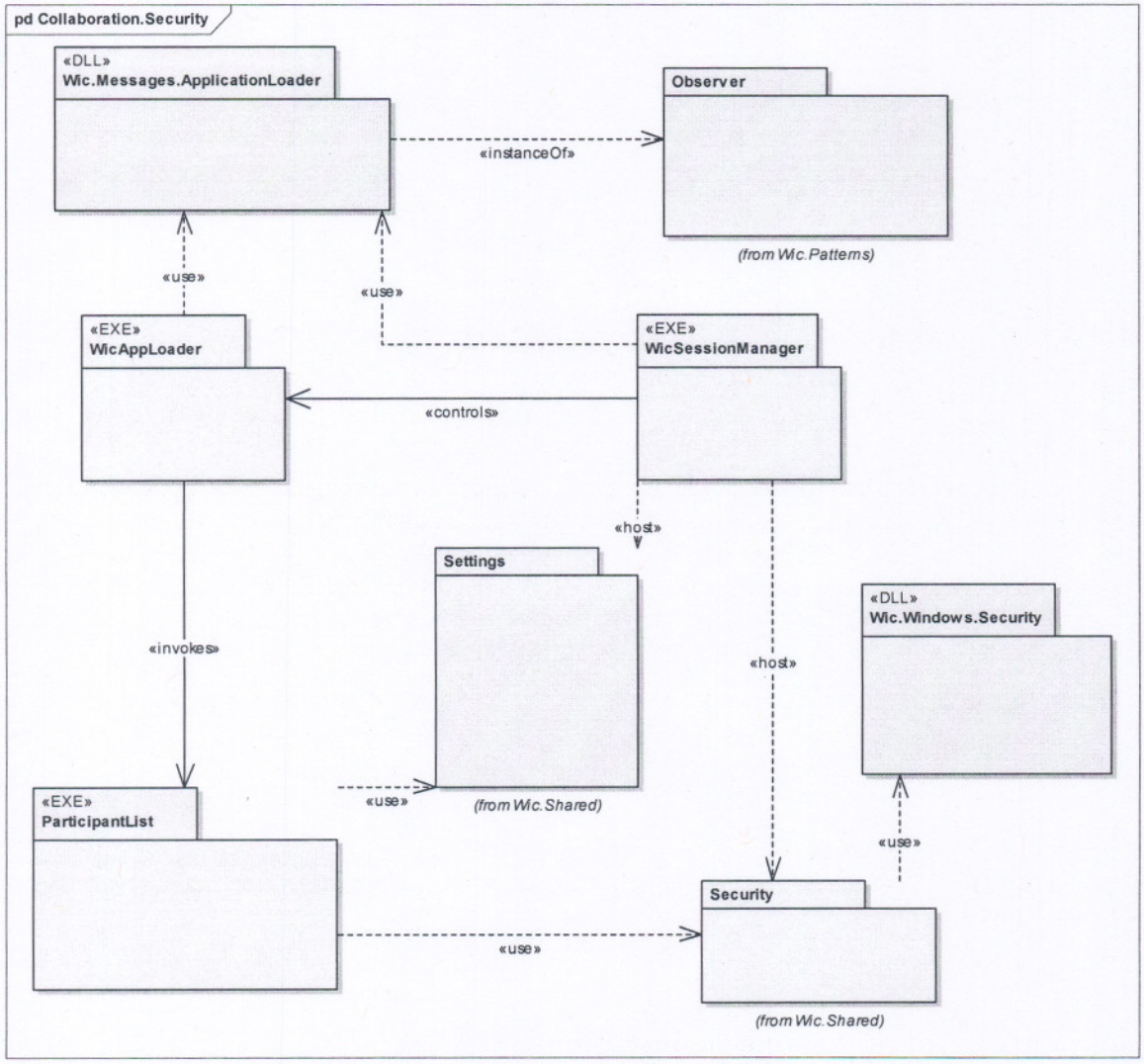


Figure 5 - WIC Session Collaborations

This relationship can be seen in the collaboration diagram in figure 5 in that the ParticipantList process uses "remoted" runtime objects in the Security package that are hosted from the WicSessionManager.

10.4 Applying Software Updates

The WicAppLoader process invokes the ApplicationUpdater process which is responsible for downloading any software updates available for the application being started. These updates are obtained by comparing an artifact version and/or date time stamp information between the local

manifest on the client and the "master" manifest located on the Web Server cluster. New components are downloaded and applied prior to starting the application.

10.5 Keeping Reference Data Synchronized

The WIC database is composed 2 types of data; transactional and static. These 2 types can be further classified into 5 specific categories; participant, vendor, financial, security and reference data. The focus of this document is in regard to the static reference data. Normally speaking, this data is used to provide referential integrity for code values, drive lookup lists, and control general (non-security related) customization options normally set at the state level and does not change on a regular basis.

Using smart client technology allows use to utilize each client machine to its fullest capability to provide an optimal solution to the customer. In this case the large amount of static reference data in the system can be stored on each client machine in a local relational database. Storing this data on each client will reduce the total number of "round trips" to the central data store (via the Service Oriented Architecture) and thereby improve the overall performance of the system.

Although this reference data is referred to as static, it does in fact change, but on a very infrequent basis. Smart client technology will be used to keep each client's local copy of the data in sync with the master copy located in the central data store.

When a WIC session is started, a Reference Data Synchronization (RDS) client process (exe) will be started as well. This process will be managed by the WIC Session Manager and will poll the central data store on startup and subsequently a set interval (TBD) throughout the session to retrieve any changes to the reference data. It is expected that the majority of the updates will occur during session startup as a background task. The polling process is in place to allow for incremental updates that may occur during the day while the session is running.

The RDS client will use a GetReferenceData web service call to retrieve any updates by passing the date and time of the last synchronization as an argument. (Note that this date and time is the date and time as understood by the Central Data Store in order to avoid any time zone collisions.) The service will return a ReferenceData collection object containing the date and time of the sync request and a heterogeneous collection of the tables that have been changed since the last sync data time. Each table in the collection will then replace the table in the local database. If no updates are needed the service will return an empty collection. In either case, the date and time of the request is then written to the local SessionSettings.xml configuration file to be used on the next request.

Note that if any record in one of the target tables has a create date time or modify date time greater than the clients date and time specified by the client the entire table is retrieved from the central data store and replaced on the client machine.

This reference data will be stored on the client machine as serialized xml files.

Note that when a client machine is operating in offline mode the session manager will not start the RDS client process since no central data store is available to poll.

During the course the update process one or more tables could be updated. If the smart client application attempts to access a table that is being updated by the RDS process the smart client

will pause momentarily until the update is complete. It should be noted that updating these tables is a momentary activity that normally takes a fraction of a second.

In general the tables that are potential candidates for consideration are ones that are not changed frequently and do not contain any participant, vendor, or financial information of a transactional nature or security related data. The list of potential candidates can be found in Appendix A of this document. (Note that list will grow as we continue through the design process.)

10.6 Processes vs. Applications

A Process [space] is a technical term used to describe a software component that has it's own runtime space allocated to it by the operating system. These processes often use libraries containing common software components. In doing so, these libraries run within that application's process space. In our case processes (like ParticipantList.exe and ParticipantFolder.exe) work together to form what is perceived as an application by the user.

For example, the ParticipantList executable performs the role of being the primary entry point for the Clinic, CAS, and State Office applications. The ParticipantFolder executable is started by the ParticipantList when the user opens a folder. More than one folder can be opened at any given time, hence the need for an independent executables.

In each case the executables are started in a different mode which controls the feature/function set that is available. (Note that access to these features and functions is then further governed by user profile that contains privileges granted to the user via the role-based security model.)

10.7 Packaging and Deployment Strategies

Processes are complied into executables (exe files) and are used to encapsulate a unique set of functionality. For example, searching for a participant is packaged into the ParticipantList executable. Libraries are complied as dynamic linked libraries (dll files) and are used to package components that are either re-used by executables and other libraries, or to package similar functionality. For example, the interfaces displayed as part of the Clinic application that are common across the PartipantList and ParticipantFolder executables are packaged into the Wic.Windows.Participant.dll. In turn this library includes Wic.ServiceAgents which contains the ParticipantServiceAgent class that provides the methods that invoke the WebService calls to the Central Data Store.

As described above, a number of software components have been grouped into applications. As an example, the system includes the Clinic, CAS, and State Office, and Vendor applications. These applications all utilize more than one executable process. These relationships have been documented in the Deployment View section of the software model and are named:

- Deployment.Clinic
- Deployment.CAS
- Deployment.StateOffice
- Deployment.Vendor

Each of these diagrams attempts to describe the inter-process relationships specific to the each for the subject applications. Additional deployment diagrams have been provided that document other processes as well address the low level dependency information for each process and related library.

Consider the deployment diagram for the Clinic application in figure 6. We can see that ParticipantList process invokes the ParticipantFolder process. These 2 processes make up the majority of the Clinic application.

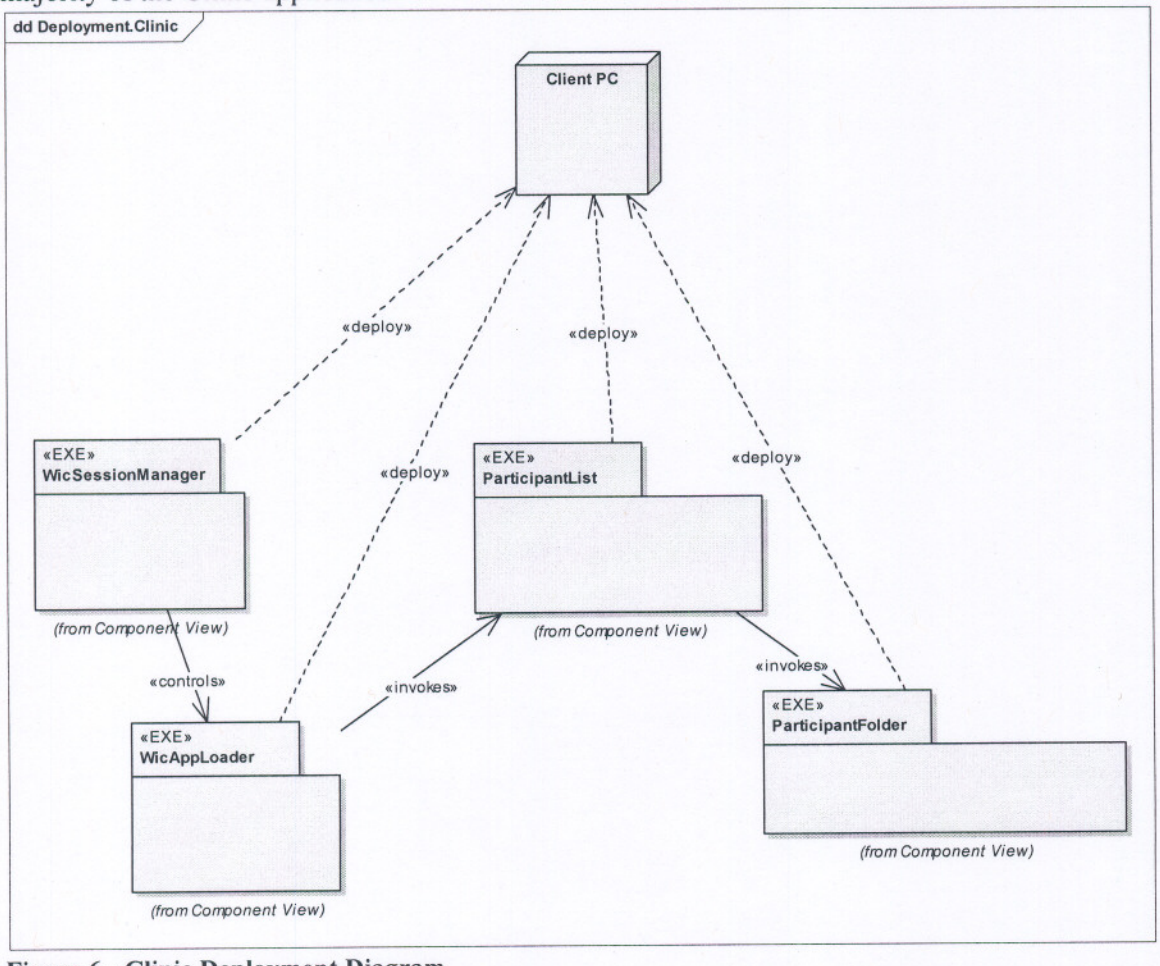


Figure 6 - Clinic Deployment Diagram

We can also determine from the diagram the WicApploader starts the ParticipantList process. The rational for using this inter process mechanism this is described in detail in the corresponding detailed documentation contained the design addendums and software model. In brief, the WicSessionManager essentially controls the session under which all of the WIC application run. The WicApploader to ensures that the session is running (indicating that proper security protocols are in place) and software any updates have been applied prior to starting the ParticipantList process.

Figure 7 illustrates the library dependencies for the ParticipantList process. This diagram can also be found in the Deployment View on the Software Model.

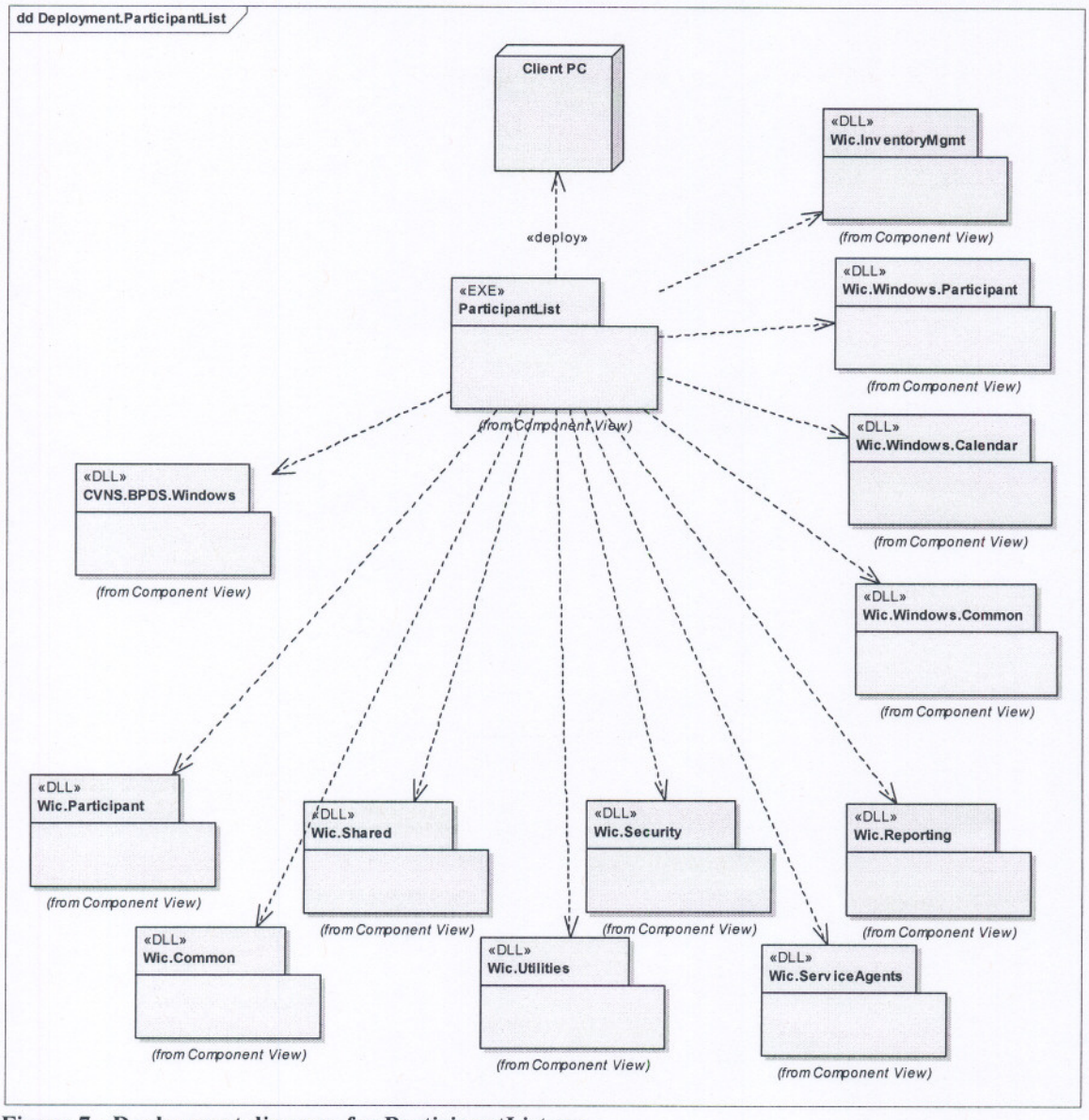


Figure 7 - Deployment diagram for ParticipantList.exe

Note that other applications, processes, and libraries are documented in a similar fashion within the Deployment View in the Software Model documentation.

The system includes the WIC Available Applications Manager (WAAM) which is used to control the initial installation of the various WIC Modules on each client PC. Subsequent to this installation all software updates to the installed applications are downloaded and applied on demand as needed by the AutoUpdater process which is started by the WicApploader process each time an application is started.

There are a number of sibling sub-folders created during installation on each client PC, two of which are; Wic and Data. The Wic folder contains all the binary files that comprise the software suite. The data folder contains a number of ASCII files used by the applications at runtime.

WIC Client Software
During installation shortcuts are created and added to the WIC Software menu located the Start > All Programs menu. There is one installation script that installs all the client side software as well as a shortcut manager that allows an authorized user to create the appropriate application shortcuts to the software. For details regarding the installation of the software please refer to the Software Installation Guide documentation.

WIC Web Server
The WIC Web Server script installs the Wic.Services component and creates the Wic.Services virtual directory.

WIC Back Office Applications
The Back Office Server script installs the EOD and EOM software components

10.7.1 Client Applications

The following applications typically reside on a client PC and provide the majority of the end user functionality contained in the system.

Clinic
The Clinic application is actually as combination of the ParticipantList.exe and ParticipantFolder.exe assemblies. The ParticipantList.exe is the primary interface for the Clinic application and launches the ParticipantFolder.exe assembly as needed. The Startup command for clinic is:

WicApploader.exe ParticipantList.exe Clinic

CAS
The Clinic application uses the same assemblies as the clinic application as well as the Master Calendar assembly, also launched from Participant list. The startup command is:

WicApploader.exe ParticipantList.exe CAS

State Office
The State Office application uses the same assemblies as the clinic application. The startup command is:

WicApploader.exe ParticipantList.exe State

Vendor Management
The Vendor Management application is actually as combination of the Vendorist.exe and VendorFolder.exe assemblies. The VendorList.exe is the primary interface for the Vendor application and launches the VendorFolder.exe assembly as needed. The Startup command for Vendor is:

WicApploader.exe VendorList.exe

Financial Management
The primary assembly for the Financial Management application is FinancialManagement.exe. The startup command is:

WicApploader.exe FinancialManagement.exe

Direct Distribution
The primary assembly for the Direct Distribution application is DirectDistribution.exe. The startup command is:

WicApploader.exe DirectDistribution.exe

Reference Utility
The primary assembly for the Reference Utility application is WicRefUtil.exe. The startup command is:

WicApploader.exe WicRefUtil.exe

Management Console
The primary assembly for the Management Console application is WicMgmtConsole.exe. The startup command is:

WicApploader.exe WicMgmtConsole.exe

System Administration
The primary assembly for the System Administration application is WicSystemAdmin.exe. The startup command is:

WicApploader.exe WicSystemAdmin.exe

Configuration Editor (Administrative tool)
The primary assembly for the Wic Configuration Editor application is WicConfigEditor.exe. The startup command is:

WicConfigEditor.exe

10.7.2 Supporting Assemblies and Utilities

The following are assemblies and utilities that are used by the various applications throughout the system based on each applications specific needs.

WicApploader
The WicApploader.exe process is used to start all WIC applications. It is responsible for ensuring that the most current application updates are downloaded and that if needed the WicSessionManager is running prior to starting the requested WIC Application. The WicApploader displays a Splash screen which is used by the WicAppLoader to provide progress information to the user during the startup process.

The WicApploader first invokes the WicAppUpdater which ensures that any applicable software updates are applied to the system.

If the requested application is a batch process then the WicApploader starts the requested process and terminates skipping the remaining activities described below.

If the requested application is not batch process then the WicApploader displays the splash screen and configures the following .NET Remoting:

- A singleton service providing a reference to an instance of the `Wic.Messages.ApplicationLoader.ApplicationLoaderSubject` class fom the WicApploader process. This is used by other processes to relay messages to the WicApploader.
- A client used to reference an instance of the `Wic.Shared.Security.AuthenticationManager` class typically provided by the WicSessionManager. This class provides the information needed by all WIC applications to authenticate during each Web Service invocation.
- A client used to reference an instance of the `Wic.Shared.Settings.SettingsManager` class typically provided by the WicSessionManager. This class provides information regarding the user that is "signed on" to the system. This information (which includes the users role based permission set) is used by each application to perform interactive authorization to the various function points in each of the WIC applications.

After configuring .NET Remoting, the WicApploader checks to see if the WicSessionManager is running. If not, the WicApploader synchronously starts the WicSessionManager, puts its main thread in a wait mode, listening for messages via the instance of the `ApplicationLoaderSubject` exposed as a singleton service via .NET Remoting. The WicApploader accomplishes this by attaching an in process instance of the `ApplicationLoaderObserver class` to the aforementioned singleton instance of `ApplicationLoaderSubject` class. This allows the WicApploader process to sit in a wait state but still receive event notifications (via callback invocations from the `ApplicationLoaderSubject` object) to the attached `ApplicationLoaderObserver` object. The WicApploader listens for 2 events from the observer; a LoadAttemptProgress event and a LoadAttemptResponded event.

The LoadAttemptProgress event is sent from the process being started (in this case the WicSessionManager) periodically and contains information that is displayed by the WicApploader on the splash screen.

The LoadAttemptResponded event is sent from the process being started (in this case the WicSessionManager) to signalthe WIcApploader that it can continue. (In the case of the WIcSessionManager this means that the user is signed on and that the instances of the `AuthenticationManager` and `SettingsManager` classes are available vis .NE Remoting. In response to receiving this event notification the WicApploader releases the wait state of its main thread and continues.

At this point the WicApploader assess the result code returned by in the LoadAttemptResponded event notification. If the result indicate success, the WicApploader asynchronously starts the requested WIC application..

Finally, the WicApploader dismisses the splash screen and terminates.

WicAppUpdater

checks the designated Web Server for any software updates. If any updates are found, they are downloaded and applied to the system.

WicSessionManager
The WicSessionManager is responsible for maintaining a local "session" containing user authentication and authorization information that can be used by any of the WIC applications via .NET Remoting.

While starting the WicSessionmanger configures the following .NET Remoting:

- A client used to reference an instance of the `Wic.Messages.ApplicationLoader.ApplicationLoaderSubject` class from the WicApploader process. The WicSessionManager uses this remoted object to relay messages to the WicApploader.
- A singleton service referencing an instance of the `Wic.Shared.Security.AuthenticationManager` class. This class provides the information needed by all WIC applications to authenticate during each Web Service invocation.
- A singleton service referencing an instance of the `Wic.Shared.Settings.SettingsManager` class. This class provides information regarding the user that is "signed on" to the system. This information (which includes the users role based permission set) is used by each application to perform interactive authorization to the various function points in each of the WIC applications.

While starting a session the process prompts for user id and password information and authenticates the user to the system.

Once the user is authenticated the WicSessionManager retrieves the user authorization information and places it in an remotable instance of the AuthenticationManager class. Subsequently, the process then attempts to update the local reference data. (Note that while checking for, and retrieving, local reference data updates the WicSessionManager displays progress window.)

The WicSessionManager uses the UpdateLocalReferenceData data web service to retrieve all tables (as xml files) that have been touched since the last time the WicSessionManager requested and update. (this date time stamp information used is stored locally on each PC and is exposed as a property of the SessionSettings class. All updates are written as xml files to the data folder specified by the SessionSettings class.

The WicSessionManager is available in the system tray while running. Selecting the Update Local Reference Data menu item from the icons context menu (right-click on the menu) forces a real-time update. Selecting the Logoff menu item closes any open wic applications and ends the session.

Wic.LocalCache.LocalReferenc Data
Local reference data refers to a number of "semi-static" lists of data that are maintained on the Central Data Store and downloaded on demand to each PC by the WicSessionManager. Local Reference Data is identified and managed via the ReferenceTableCatalog table in the database. This table contains a list of all tables in the database that contain data that can be treated as local reference data (i.e. can be downloaded to each PC as an xml file). Each table referenced in this

table has an insert/update/delete trigger that modifies the ModifyDttm attribute of the corresponding row in the ReferenceTableCatalog to reflect the last date and time the table was "touched". The following as an example of one of these triggers:

```
CREATE TRIGGER [dbo].[tgr_wicstatus_iud]
ON dbo.WICSTATUS
FOR INSERT, UPDATE, DELETE
AS
UPDATE REFERENCETABLECATALOG SET MODIFYDTTM = GETDATE() WHERE
REFERENCETABLECATALOGID = 'WICSTATUS'
```

Local reference data is accessed by the software via an instance of the Wic.LocalCache.LocalReferenceData class which provides an enumeration of known list of datasets that are available as local reference data.

Wic.LocalCache.Dictionary
Provides a customized interface to the ReferenceDictionary.xml file via the LocalReferenceData class. This customized interface is required since the data contained in the xml file is of a heterogeneous nature.

Wic.LocalCache.StateBusinessRules
Provides a customized interface to the StateBusinessRules.xml file via the LocalReferenceData class. This customized interface is required since the data contained in the xml file is of a heterogeneous nature.

Wic.Common.SessionSettings
The Wic.Common.SessionSettings class provides an interface to the SessionSettings xml file is used to store session level information on each client PC. The SessionSettings.xml file must be located in the same folder as the binaries attempting to access it via the SessionSettings class.

Tag names used to identify sensitive information are obfuscated using a nondescript naming convention for additional security. For clarity sake following obfuscation is being used:

T1 contains the data source
T2 contains the database name
T3 contains the database user id
T4 contains the database password

CVNS.BPDS.Logger (Event Logging Utility)

Logger is component of the CVNS.BPDS Framework that provide for a consistent method of retaining pertinent application information and events. The retention mechanism used is a text file whose file name is the current date expressed as YYYYMMDD with a .log extension appended to it. This provides us with a daily log file and provide for easier archival and cleanup of historical logs. Note that this archival/cleanup mechanism is the responsibility of the individual maintaining the machine and is not provided by the framework.

Following the standard set by Microsoft these text files are created at:

```
%SystemRoot%/system32/Logfiles/CVNS.BPDS
```

where %SystemRoot% is typically c:\Windows. Each entry in the log file contains: a timestamp, the full path to the executable process that made the entry, and the actual data to be logged. Each of these elements is separated by a colon.

Work supports four types of information that can be written to the log file. They are; Software Updater progress and results, a stack trace for all exceptions, DataSync results, and optionally all of the sql statements sent to the database. Further details on the information logged for each of these is outlined below.

Software Updater
The Software Updater logs the following information:
The start of the process, including the area that is being updated
The number of files and assemblies that need to be updated
The successful download of each artifact including the date time stamp for files and the version number for assemblies
The execution of any Post Update commands
The completion status of the process
Any and all errors that might occur

Stack trace for all exceptions
All exceptions that are handled by the function HandleException that is provided by the CVNS.BPDS.Windows.Forms.Form class be logged with a full stack trace.

DataSync results
The DataSync application logs it's progress and any errors that occur.

Sql Statements
While the logging in each of the scenarios mentioned above is automatically enabled and cannot be disabled, the logging of SQL Statements is disabled by default and must explicitly be enabled by adding an appsetting to the appropriate .config file. In the case of on-line clients it must be enabled in the web.config on the central web server. For off-line clients it must enabled in each and every app.config file whose sql you wish to log. This entry is: <add key="CVNS.BPDS.Logger.Level" value="Info" />

10.7.3 Web Server Configuration

The Web Server runs 13 sets of Virtual Directories; one for each ITO, these are also referred to as Sites. Each Site has the following configuration:

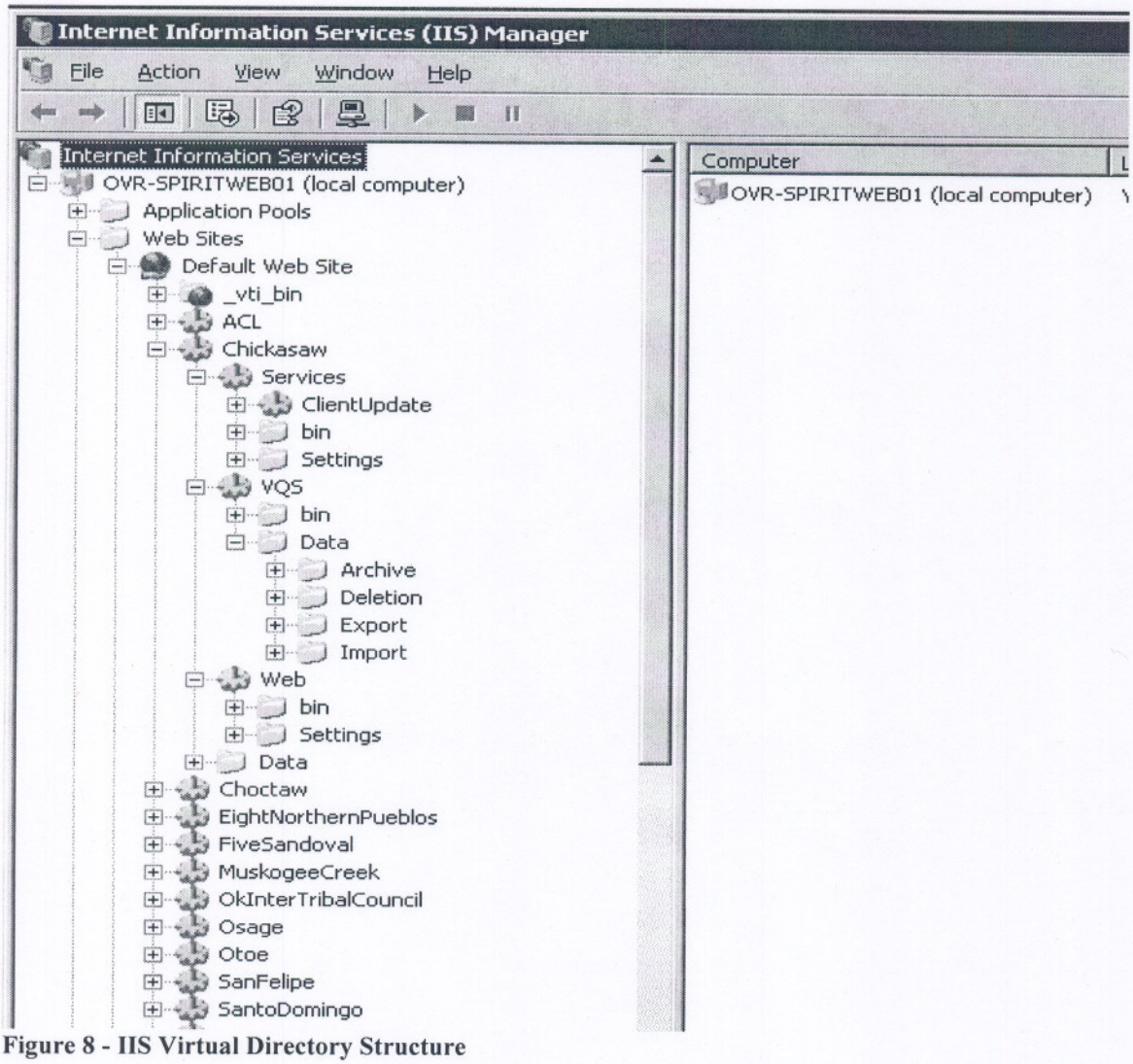


Figure 8 - IIS Virtual Directory Structure

The Services directory is "pointed to" the physical directory C:\WIC\Sites\[ITO]\Services where [ITO] is the name of the ITO. This physical directory contains all the assemblies and configuration files needed to publish the specified ITO's web services.

The VQS directory is "pointed to" the physical directory C:\WIC\Sites\[ITO]\VQS where [ITO] is the name of the ITO. This physical directory contains all the assemblies and configuration files needed to publish the ITO's Vendor Questionnaire System web application. (Online Vendor Applications and Price Surveys)

The Web directory is "pointed to" the physical directory C:\WIC\Sites\[ITO]\Web where [ITO] is the name of the ITO. This physical directory contains all the assemblies and configuration files needed to publish the ITO's Participant Questionnaire web application.

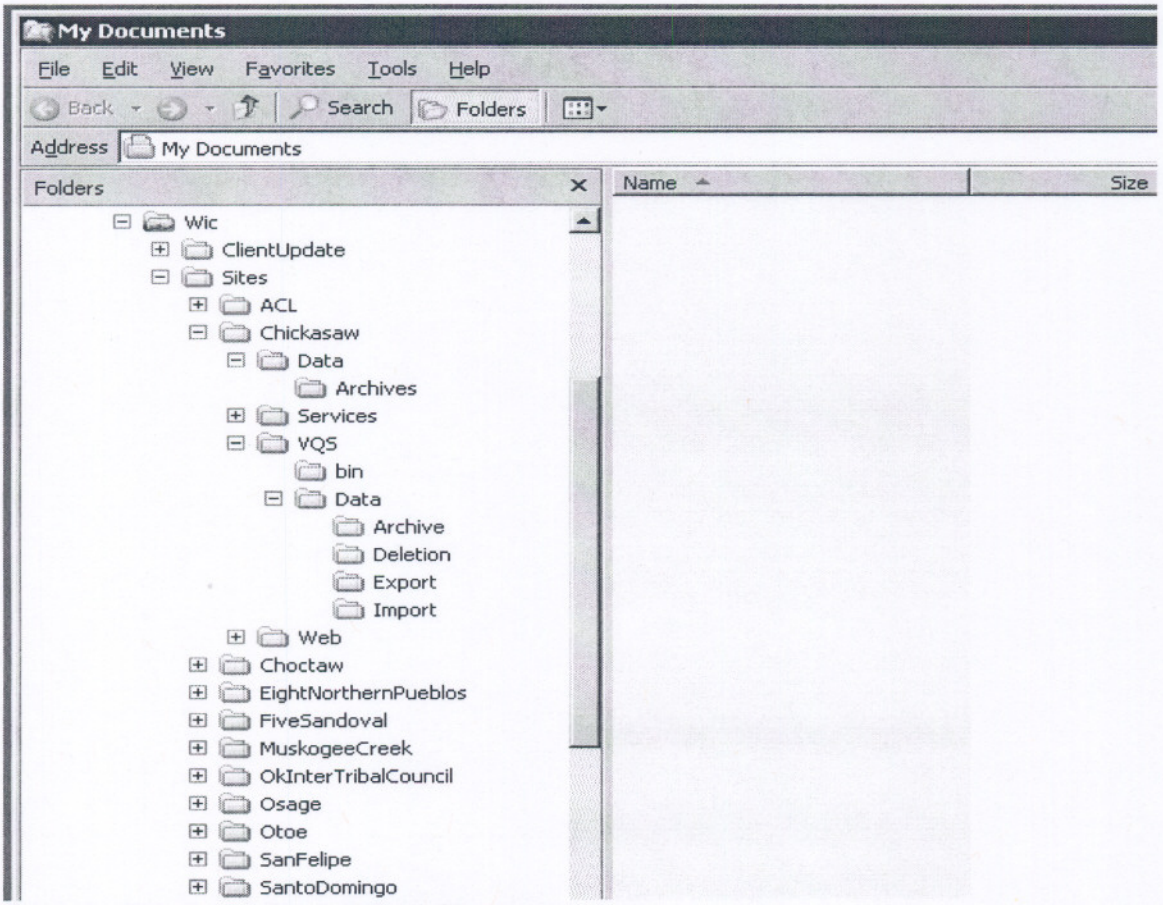


Figure 9 - Web Server Physical Folder Structure

The VQS Folder in each ITO's folder hierarchy contains a number of sub-folders that are used to exchange data between the VQS Web Application and the ITO's SPIRIT System. In order to facilitate this file exchange there are 2 Windows Services running on the Web Server; VQS_SurveyImport and VQSSurveyProcessor..

Figure 10 - Installation folder for the Windows Services

The Vqs_SurveyImport service is a component of the VQS Web Site that monitors specified folders for the arrival of xml documents that are to be used to publish and/or send invitations to surveys. The service is installed in C:\Wic\WindowsServices\bin. This service uses a configuration file named VQS_SurveyImport.xml which contains site information detailing the list of folders to be monitored as well as the database connection information to be used to store the processed xml documents. The VQS_SurveyImport service does not use web services to communicate with the database. Since the service resides on the web server and is in close proximity to the database server a SqlClient connection is established directly with the database in order to eliminate any unnecessary traffic via the web services. Below is a sample entry from the VQS_SurveyImport.xml file:

```
<Sites>
    <Site  name="Chickasaw">
        <dbType>SQLSERVER</dbType>
        <dbServer>ls;gdjor</dbServer>
        <dbSchema>flgjsvojs[v</dbSchema>
        <dbUserID>sldjosdfg</dbUserID>
        <dbPassword>sobjssos</dbPassword>
        <UserId>gpweirhq</UserId>
        <WebUrl>http://localhost/Chickasaw/VQS_Web/</WebUrl>
        <VqsFtpSite>c:\Wic\Sites\Chickasaw\VQS\Data</VqsFtpSite>

    <VqsSurveyTemplates>c:\Wic\Sites\Chickasaw\VQS\VQS_SurveyTemplates</VqsSurveyTemplates
>
        <VqsSchemas>c:\Wic\Sites\Chickasaw\VQS\VQS_Schemas</VqsSchemas>
        <FromEmailAddress>vqsweb@chickasaw.com</FromEmailAddress>
        <SmtpServer>172.16.1.254</SmtpServer>
    </Site>
</Sites>
```

The VQSSurveyProcesor service is component of the SPIRIT system that monitors specified folders to the arrival of xml documents containing responses to surveys that have been posted to the VQS web site. The service is installed in C:\Wic\WindowsServices\bin. This service uses a configuration file named VQSSurveyProcessor.xml which contains site information detailing the list of folders to be monitored as well as the database connection information to be used to store the processed xml documents for each ITO. The VQSSurveyProcessor service does not use web services to communicate with the database. Since the service resides on the web server and is in close proximity to the database server a SqlClient connection is established directly with the database in order to eliminate any unnecessary traffic via the web services. Below is a sample entry from the VQSSurveyProcessor.xml file:

```
<Sites>
    <Site  name="Chickasaw">
        <SmtpServer>172.16.1.254</SmtpServer>
        <SettingsPath>c:\Sites\Chickasaw\Services</SettingsPath>
        <MonitorPath>c:\Sites\Chickasaw\VQS\Data\Export</MonitorPath>
        <ArchivePath>c:\Sites\Chickasaw\Data\Archives</ArchivePath>
        <UserId>FGOjJW/DiOQvxQrYOodZhg==</UserId>
        <Password>FGOjJW/DiOQvxQrYOodZhg==</Password>

    <EmailNotificationList>rnash@covansys.com;thumphries@covansys.com</EmailNotificationLi
st>
        <EmailSender>VqsSurveyProcessor@spirit.org</EmailSender>
    </Site>
</Sites>
```

Notice that in this configuration the VQS_SurveyImport service watches the Import of each ITO VQS web site (C:\Wic\[ITO]\VQS\Data\Import; the value of the VqsFtpSite element is used in conjunction with the Import folder which is coded into the software) while the VQSSurveyProcessor service watches the Export folder of each ITO VQS web site (value of the MonitorPath element).

This mechanism must be in place because the VQS web site can be configured as a standalone website and database that uses FTP to exchange data. In this installation however, the VQS website and the SPIRIT system are installed on the same server and use the same database therefore the exchange of files is facilitated by simply placing the files in the designated folders thereby circumventing the use of FTP.

To manually install the processors as a windows services open a cmd window and run the following command from the folder containing the executable:

C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\installutil /i [ProcessorExe].
(Where [ProcessorExe] is the name of the exe to be installed as a windows service.)

Once completed start the service by using the Web Server's Computer Management application (right click on My Computer and select Mange then expand Services and Applications and select the Services item. Select the process name and press the start button in the toolbar.

To stop the service use the Web Server's Computer Management application (right click on My Computer and select Mange then expand Services and Applications and select the Services item. Select the process name and press the stop button in the toolbar.

To manually uninstall the windows service, stop the service (see above) then open a cmd window and run the following command from the folder containing the executable:

C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\installutil /u [ProcessorExe].
(Where [ProcessorExe] is the name of the exe to be uninstalled.)

10.7.4 Application Server Configuration

The Application Server is used to host the End-of-Day and End-of-Month processing. Similar to the Web Server, a folder structure in in place for each of the 13 ITO's; C:\WIC\Sites\[ITO] where [ITO] is the name of the ITO. Both the EODProcess and EOMProcess executables reside in the ??? folder and used to run the respective processes for each ITO. This is accomplished by passing the name of the ITO as a command line argument when starting the process. Where name is the value of the name attribute found the EODProcess.xml file. Note that both processes use the EODProcess.xml configuration file. Below is a sample site entry from the EODProcess.xml file.

```
<Sites>
    <Site name="Chickasaw" >
        <ProcessId>WICEOD</ProcessId>
        <BankingSystemName>BANKING</BankingSystemName>
        <CdcSystemName>CDC</CdcSystemName>
        <StateSystemName>STATE</StateSystemName>
        <UserId>FGOjJW/DiOQvxQrYOodZhg==</UserId>
        <Password>nXZy/kQTMEOlIw8C22nsQQ==</Password>
        <SmtpServer>172.16.1.254</SmtpServer>
        <EmailSender>eodprocess@chickasaw.net</EmailSender>
        <SettingsPath>c:\WIC\Sites\Chickasaw</SettingsPath>
        <CdcFolder>c:\WIC\Sites\Chickasaw\Data\EOD\Cdc</CdcFolder>
        <BankingFolder>c:\WIC\Sites\Chickasaw\Data\EOD\Banking</BankingFolder>
        <VendorFolder>c:\WIC\Sites\Chickasaw\Data\EOD\Vendor</VendorFolder>
        <EomReportsFolder>c:\WIC\Sites\Chickasaw\Data\EOM\Reports</EomReportsFolder>
        <LogFileFolder>c:\WIC\Sites\Chickasaw\Data\EOD</LogFileFolder>
        <DefaultLogFileName>LogFile.txt</DefaultLogFileName>
    </Site>
</Sites>
```

Notice the SettingsPath element that references the location of ITO's SessionSettings.xml file where the database connection information resides. Note that the ProcessId and various system name elements must match entries in the COMMREQUEST, EMAILRECIPIENT, and EMAILCONTENT tables in the ITO's database. The UserId and Password elements must contain encrypted values for a user in the USERPROFILE table of the ITO's database.

The EOD Process can connect via FTP to an external banking contractor, the CDC, and/or a designated State entity for the purposes of sending and receiving files. This communication information is stored in the COMMREQUEST table. The various system name elements in the configuration information stipulate the SYSTEMNAME value to be used when retrieving the appropriate record from the COMMREQUEST table. IN the sample above the communication information for the Banking contractor can be found on the WICEOD/BANKING record in the COMMREQUEST table of the ITO's database.

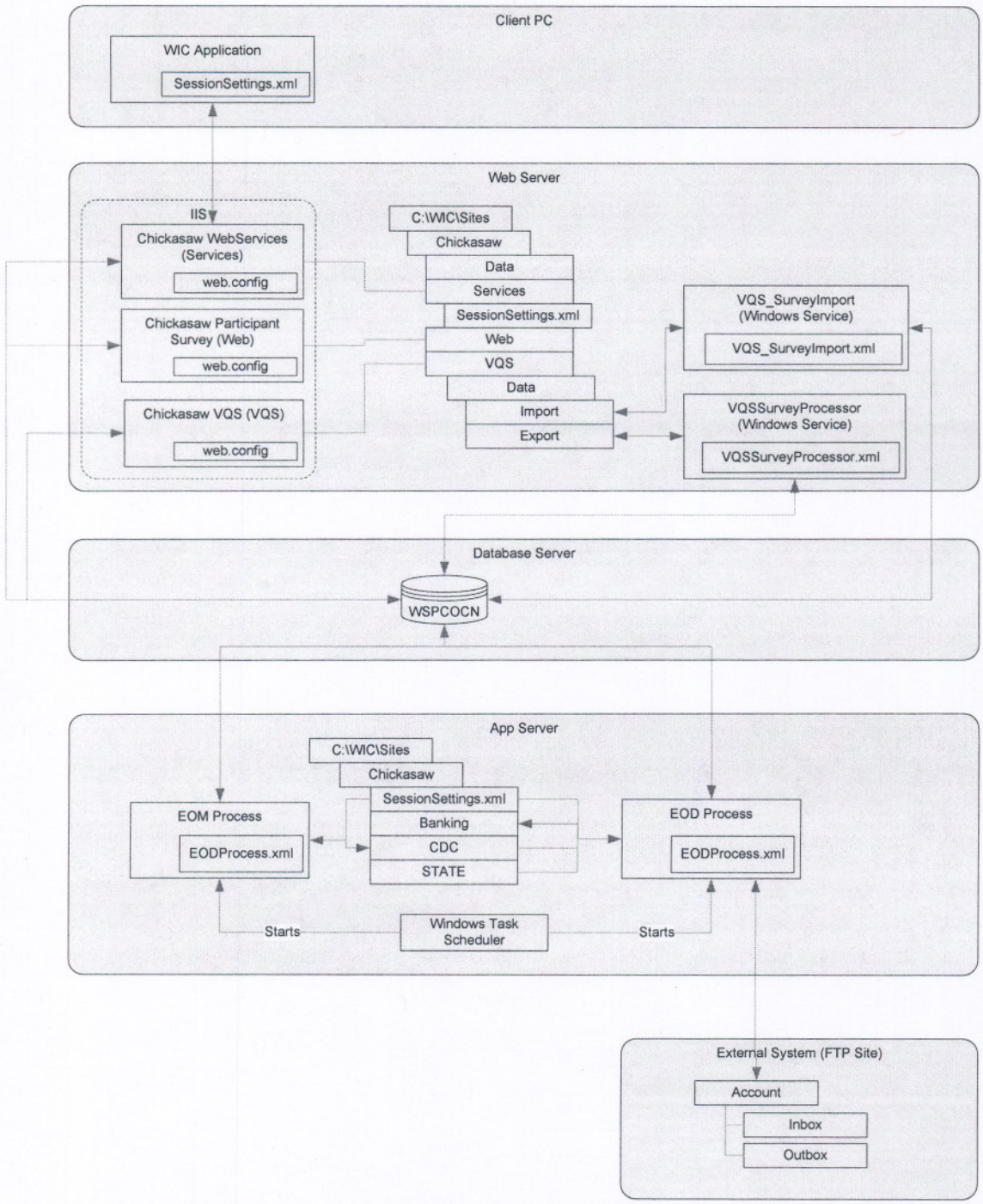
Note that folders referenced in the COMMREQUEST records must exist in the Application Server. In general the following rules should be followed:

- If files are to be send and/or received from an external banking contractor
    - Ensure that a WICEOD/BANKING record is in the COMMREQUEST table.
    - Ensure that the BankingSystemName element in the EODProcess.xml file has the same value as the SYSTEMNAME field of the corresponding banking record in the COMMREQUEST table. Typically this value is 'BANKING'
    - Ensure that the values of the INBOUNDFOLDER and OUTBOUNDFOLDER fields in the WICEOD/BANKING Record in the COMMREQUEST table refer to the same path as the value specified in the BankingFolder element of the EODProcess.xml file.

  - Ensure that paths referenced in the INBOUNDPOSTFOLDER and OUTBOUNDPOSTFOLDER exist on the Application Server.
- If files are to be end to the CDC (Note that we do not support receiving files from the CDC)
  - Ensure that a WICEOD/CDC record is in the COMMREQUEST table.
  - Ensure that the CdcSystemName element in the EODProcess.xml file has the same value as the SYSTEMNAME field of the corresponding cdc record in the COMMREQUEST table. Typically this value is 'CDC'
  - Ensure that the value of the OUTBOUNDFOLDER field in the WICEOD/CDC Record in the COMMREQUEST table refers to the same path as the value specified in the CdcFolder element of the EODProcess.xml file.
  - Ensure that paths referenced in the INBOUNDPOSTFOLDER and OUTBOUNDPOSTFOLDER exist on the Application Server.
- If participation files are to be sent to another State Agency, e.g. State of Oklahoma. (Note that we do not support receiving files from other state agencies)
  - Ensure that a WICEOD/STATE record is in the COMMREQUEST table.
  - Ensure that the StateSystemName element in the EODProcess.xml file has the same value as the SYSTEMNAME field of the corresponding state record in the COMMREQUEST table. Typically this value is 'STATE'.
  - Ensure that paths referenced in the INBOUNDPOSTFOLDER and OUTBOUNDPOSTFOLDER exist on the Application Server.
- If any of the COMMREQUEST records have the SENDEMAIL field set to 'Y' then ensure that there are corresponding entries in the EMAILRECIPIENT and EMAILCONTECT tables.

Server Configuration Overview

The following diagram depicts the various servers, processes, and file system structures in place for the SPIRIT system.



Starting the client-side applications

All client-side WIC applications are started using the WicApploader. In order to start an application you would use the following syntax from a command line.

```
WicApploader.exe [processname] [args]
```

[processname] is the name of the exe to be started.

[args] are the arguments (if any) to be passed to the exe being started

For example:
The following command line invocation will start the participantList application in Clinic mode:

```
WicApploader.exe ParticipantList.exe Clinic
```

Starting the batch processes
The EOD and EOM processes are typically started by the windows task scheduler using the following syntax.

```
EODProcess.exe [site]
EOMProcess.exe [site]
```

Where [site] is the name of the ITO as listed in the name attribute of the Site element in the EODProcess.xml file which is located in the same folder as the EODProcess.exe and EOMProcess.exe files. (Note that both exe use the same xml configuration file.)

10.7.5 Packaging Overview

The following list will serve as a high level overview of the various packages included as part of the system.

Package	Type	Description
DataSyncClient	EXE	Process used to control checking out and checking in clinic data.
DirectDistribution	EXE	Process used to record redemption of food benefits in a direct distribution environment. Primary exe of the Direct Distribution application/module
EODProcess	EXE	End-of-Day processes. Run as scheduled tasks on the batch/application server
EOMProcess	EXE	End-of-Month processes. Run as scheduled tasks on the batch/application server
FinancialManagement	EXE	Process used to manage financial information such as journals and rebates. Primary exe of the Financial Mgmt application/module
MasterCalendar	EXE	Process used to manage clinic working hours and appointment resources. Secondary exe of the CAS application/module
ParticipantFolder	EXE	Process used to manage information regarding a participant. Acts as a gateway to the certification process (Wic.CertGuidedScript) Secondary exe of the Clinic, CAS, and State Office applications/modules
ParticipantList	EXE	Process used to search for participants. Acts as a gateway to the Participant Folder (ParticipantFolder). Primary exe of the Clinic, CAS, and State Office applications/modules.
ReportGenerator	EXE	Process used to create, run, print, and store adhoc queries
Reports	N/A	Repository of crystal report templates used by the system
ScheduleJobAdmin	EXE	Process used to manage the scheduling of the EOD and

		EOM processes
VendorFolder	EXE	Process used to manage information regarding a vendor. Secondary exe of the Vendor Mgmt application/module.
VendorList	EXE	Process used to search for vendors. Acts as a gateway to the VendorFolder (VendorFolder). Primary exe of the Vendor Mgmt application/module.
VQSSurveyProcessor	EXE	Windows Service used to process incoming survey response documents (e.g. Online Vendor Applications and Vendor Proce Surveys)
WAAM	EXE	WIC Available Applications Manager. Process used to perform the initial installation of the Client WIC applications. (E.g. Clinic, State Office, CAS, Vendor Mgmt, etc.)
WicApploader	EXE	Process used to launch all other WIC applications/modules. Ensures that a session is running (WicSessionManager) and that all applicable software updates have been applied (CVNS.BPDS.AutoUpdater)
WicConfigEditor	EXE	Utility used to configure the various session settings for both the client and server applications.
WicMgmtConsole	EXE	Process used to manage user profiles, roles, permissions, and remote machine profiles.
WicRefUtil	EXE	Process used to manage the various types of reference data used by the system (E.g. Default food packages, food items, counties, risk factors, etc.)
WicSessionManager	EXE	Process that provides for a single sign-on environment on each client PC.
WicSystemAdmin	EXE	Process used to manage system-wide data relationships (E.g. (outreach) programs and organizations, state and local use questions, etc.)
Wic.Web.Questionnaire	DLL	ASP.NET web Application used to gather information from participants
Wic.Windows.Breastfeeding	DLL	Library containing user interface components specific to the breastfeeding namespace.
Wic.Windows.Calandar	DLL	Library containing user interface components specific to the calendar and appointment maintenance
Wic.Windows.CaseloadMgmt	DLL	Library containing user interface components specific to caseload management
Wic.Windows.Charting	DLL	Library containing user interface components specific to displaying Growth Charts.
Wic.Windows.Common	DLL	Library containing common user interface components used across several assemblies
Wic.Windows.FoodInstruments	DLL	Library containing user interface components specific to food packages, prescriptions, and benefits issuance.
Wic.Windows.Inventory	DLL	Library containing user interface components specific to general inventory management.
Wic.Windows.Participant	DLL	Library containing user interface components specific to managing participant information.
Wic.Windows.Reporting	DLL	Library containing user interface components specific to producing reports.
Wic.Windows.Security	DLL	Library containing user interface components related to the management of security related features. (E.g. user profiles, roles, privileges, etc)
Wic.Windows.Vendor	DLL	Library containing user interface components specific to managing vendor information.
Wic.CertGuidedScript	DLL	Library containing user interface components specific to the participant certification process
Wic.AgencyOutreach	DLL	Library containing business service components specific to the management of outreach agency information.
Wic.BatchProcess	DLL	Library containing business service components specific to the execution and management of batch processes (E.g. EOD, EOM)
Wic.Breastfeeding	DLL	Library containing business service components specific to the breastfeeding namespace
Wic.Calendar	DLL	Library containing business service components specific to managing calendars and appointments
Wic.CaseloadMgmt	DLL	Library containing business service components specific to caseload management
Wic.Charting	DLL	Library containing business service components specific to

		growth charts and statistical growth information
Wic.Common	DLL	Library containing common business service components used by several assemblies.
Wic.DataSynchronization	DLL	Library containing business service components specific to checking out and checking in participant data
Wic.DirectDistribution	DLL	Library containing business service components specific to tracking the redemption of food benefits in a direct distribution environment
Wic.FinancialManagement	DLL	Library containing business service components specific to financial management
Wic.FoodInstruments	DLL	Library containing business service components specific to food packages, prescriptions, and benefits issuance.
Wic.Inventory	DLL	Library containing business service components specific to general inventory management
Wic.LocalCache	DLL	Library containing common business service components used to manage and provide access to localize system information to the client
Wic.MailMerge	DLL	Library containing business service components specific to merging system data to MS Word documents
Wic.Messages.ApplicationLoader	DLL	Library containing business service components specific to the use of .NET Remoting.
Wic.Participant	DLL	Library containing business service components specific to the management of participant data
Wic.Patterns	DLL	Library containing interface patterns implemented in several assemblies
Wic.Reporting	DLL	Library containing business service components specific to generating reports.
Wic.Security	DLL	Library containing business service components related to the in memory management of secure information.
Wic.ServiceAgents	DLL	Library containing proxy services used between the client and web service processes for those applications that can operate in both a connected and disconnected environment.
Wic.Services	DLL	ASP.NET assembly containing the web service interfaces used by the client applications.
Wic.Shared	DLL	Library containing (.NET) remotable classes shared across the various client applications.
Wic.Utilities	DLL	Library containing business service components generic to the overall system
Wic.Vendor	DLL	Library containing business service components specific to vendor management.
CVNS.BPDS.DataAccess	DLL	Library containing the generic ANSI compliant implementation of the OLEDB Data Access Layer
CVNS.BPDS.DataAccess.SqlServer	DLL	Library containing the MS SQL Server specific implementation of the CVNS.BPDS.DataAccess library
CVNS.BPDS.Downloader	DLL	Library containing components used to manage the exchange of files from one host to another
CVNS.BPDS.Devices.SignatureCapture	DLL	Library providing a generic interface over the Topaz signature device driver
CVNS.BPDS.Logger	DLL	Library providing generic application logging services
CVNS.BPDS.Security	DLL	Library providing a set of cryptographic services that can be used to encrypt/decrypt and hash sensitive data
CVNS.BPDS.Updater	DLL	Library providing a workflow that allows for software updates using a software manifest
CVNS.BPDS.Windows	DLL	Library providing a set of base classes and utilities that can be used to for error handling at the user interface layer.

A dependency matrix has been included, in the form of an Excel spreadsheet, depicting the various assembly level dependencies of the software.

10.8 Storing the Data

Each ITO will have there own database instance. All of the databases will "run" on a single SQL Server Active/Passive Cluster. Each of these databases is considered to be the "Central Data Store" for the ITOs. The system uses a large amount of reference data to populate selection lists,

provide code translations, and maintain foreign key relationships. This reference data is stored not only in the central data store but on each client machine as well. The reference data is stored in the form of xml files and is kept in sync with the central data store via the RDS (Reference Data Synchronization) process which is controlled by the WicSessionManager process. The RDS process is run at the beginning of each session as well as throughout the duration of the session on a regularly scheduled interval. Having the reference data on each client machine reduces the overall number of round trips to the central data store as well as the total size of the service responses containing reference data code values. The RDS process uses the Service Oriented Architecture to perform the synchronization.

A number of client machines will be capable of checking out clinic data and operating in an off-line capacity. These machines will each have a SQL Server database to store this data.

The process steps related to checking data in and out from the central data store are fully documented in Chapter 11 of the Application Administration DFDD. The mechanics supporting this process use the same Service Oriented Architecture to perform the synchronization.

11 Software Design Specifications

The detailed technical design of the software is expressed using a number of UML techniques. There are 2 primary sets of detailed designs; the software model and the data model.

11.1 Software Modeling

The software model is documented using Enterprise Architect and is published in HTML format. The model contains a number of project artifacts that describe in detail the structure and collaborations of the various software components of the system. The following sections provide a brief description of each artifact type.

11.1.1 Detailed Design Addendums

Detailed Design Addendum documents are used to provide a map from the DFDD to the Object Model. Each addendum contains a table that cross references each interface described in the DFDD to the corresponding software artifact to be implemented. The addendum also contains a listing of all the sequence diagrams used to describe the various runtime behaviors described in the DFDD. These documents can be found in the Software Model/Detailed Design Addendums folder and are named by Chapter in accordance with the DFDD.

11.1.2 Activity/Data Flow Diagrams

A number of activity diagrams have been included in the software model. These diagrams are used to illustrate the control flow and datastore relationships amongst various business processes.

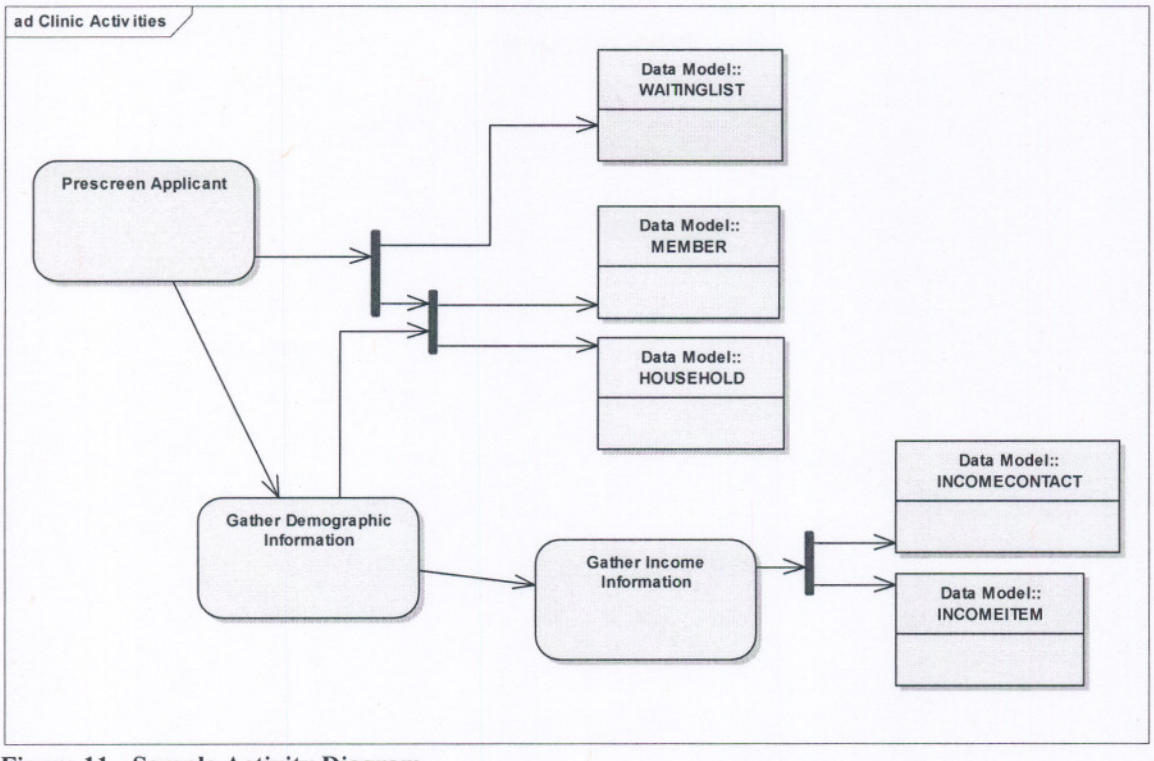


Figure 11 - Sample Activity Diagram

The diagram in figure 8 is a small portion of the Clinic Activities diagram located in the Activities section of the Dynamic View in the software model. This portion of the diagram indicates that while prescreening an applicant, information may be written to the WaitingList, Member, and Household tables. Subsequent to the prescreening the additional demographic information may be collected which in Income information may be collected. If that is the case then the income information is written to the IncomeContact and IncomeItem tables.

Additional activity diagrams can be found in the Activity section of the Dynamic View in the Software Model. (Note that a number of activity diagrams were created before the use of the Enterprise Architect modeling tool. These diagrams are being migrated into the tool and can be found in the Activity Diagrams folder on the CD.)

11.1.3 Communication Diagrams

Communication diagrams are used to depict the navigation paths and interface relationships across the various executables and libraries. These diagrams can be found in each package in the Component View section of the Software Model and typically have the word 'Navigation' in the diagram name.

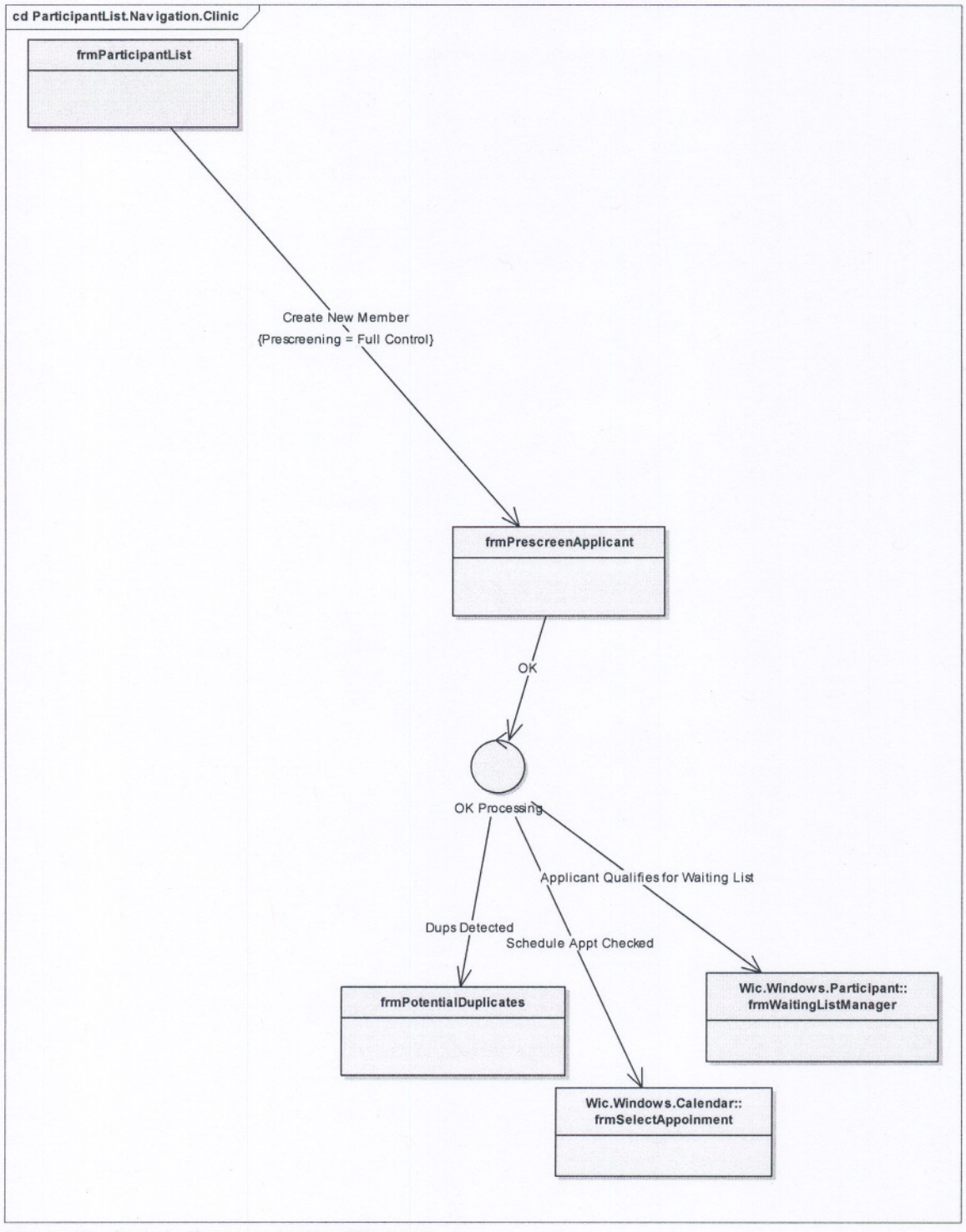


Figure 12 - Sample Communication Diagram

The sample diagram above indicates that the user interface that supports the prescreening activity is invoked from the ParticipantList executable.

11.1.4 Sequence Diagrams

Sequence diagrams are used to indicate interaction and control flow between runtime objects. These diagrams can be found in the each package in the Component View section of the Software Model. Primary diagrams are referenced in the design addendum documents.

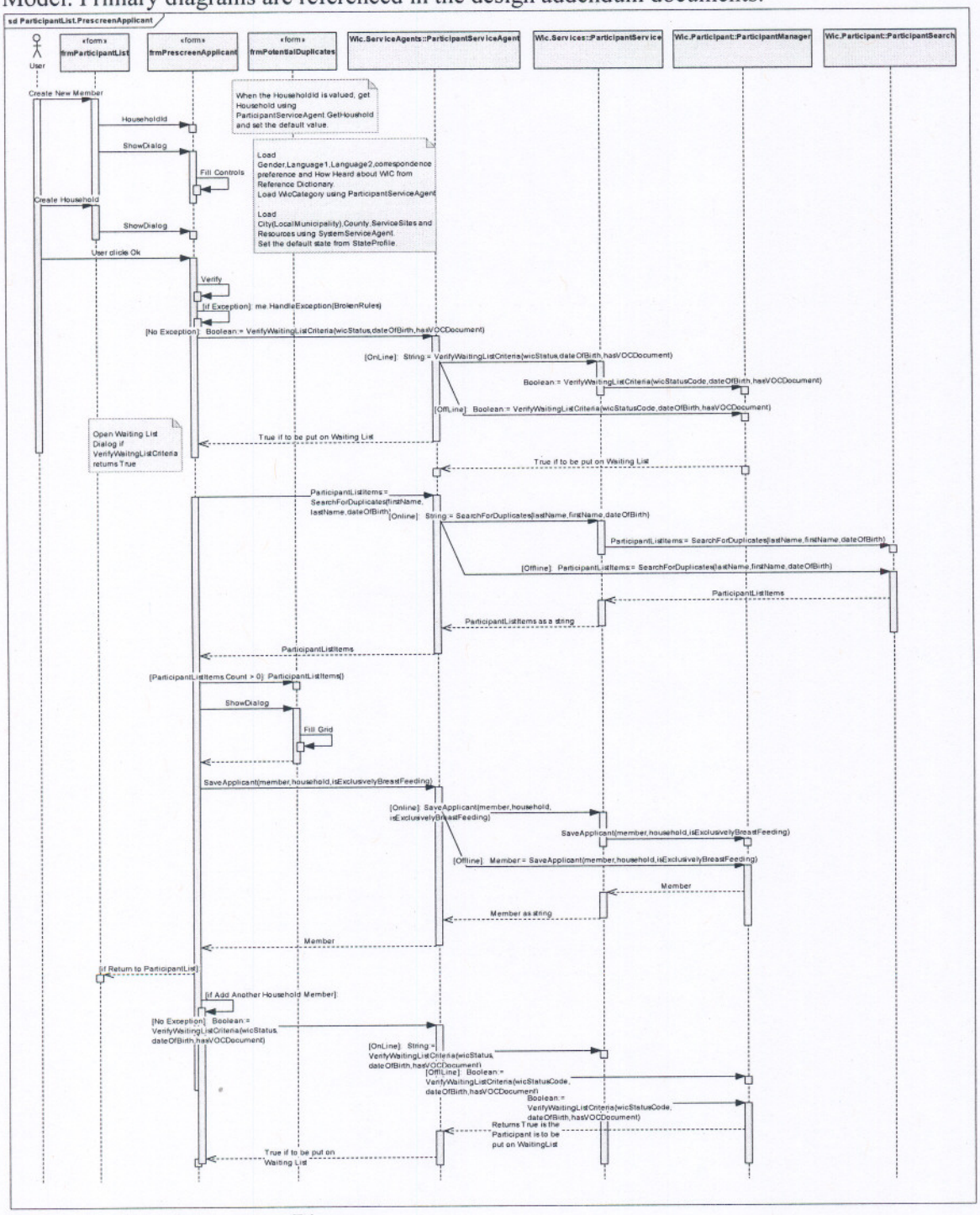


Figure 13 - Sample Sequence Diagram

The sequence diagram above describes the runtime interactions of the software components that are needed to support the prescreening activity.

11.1.5 Class Specifications

Class specifications are used to provide detailed instructions in regard to construction of classes. These specifications are used ensure that the class provides the properties, methods, and events needed to collaborate with other software objects as expressed in the aforementioned diagrams. These diagrams can be found in each package in the Component View section of the Software Model.

11.1.6 Deployment Diagrams

Deployment diagrams are used to describe the target environment and dependent relationships of the various software components. These diagrams can be found in the Deployment View section of the Software Model.

11.2 Data Modeling

The data model is documented using Power Designer and is published in HTML format. It contains the entity relationship and detailed table definitions for the underlying data model. This includes but is not limited to diagrams depicting relationships, detailed table (and view) specifications citing data types and lengths as well as table/column usage descriptions, constraints, stored procedures, and triggers.

12 Design Patterns and Practices

A number of design patterns and programming practices are to be followed during the development lifecycle of the project. The following sections document these patterns and practices. In all cases the coding practices outlined in the VB.NET Coding Guidelines document guide supersede syntax and formatting example code provided in this document. All the source code contained in this document can be found in the prototype solution named SPIRITNAV located in the Subversion repository. The SPIRITNAV solution is collection of projects used to illustrate the architecture and code that will be used during software construction. It is not meant to replace design level documentation and is superseded by any future documentation provided by the design team.

12.1 User Interface Layer

Windows forms will be used to interact with the user. Forms will be implemented as either windows or dialogs. The BPDS Windows UI Guidelines document will be used as the basic standard upon which all Spirit user interfaces will be constructed. Any exceptions to these guidelines are noted below.

12.1.1 Windows

All Windows should inherit from CVNS.BPDS.Windows.Forms.Form.
The following property settings will be applied to all windows unless otherwise stated.

FormBorderStyle	FixedSingle	
StartPosition	CenterScreen	
MinimizeBox	True	
AcceptButton	[the accept/default button for the form]	

HaloColor	`System.Drawing.SystemColors.Highlight`	
CancelButton	[the cancel button for the form]	

All other properties of windows will be left to their default values unless otherwise stated.

12.1.2 Dialogs

All Dialogs should inherit from CVNS.BPDS.Windows.Forms.Form.
The following property settings will be applied to all dialogs unless otherwise stated.

AcceptButton	[the accept/default button for the form]	
CancelButton	[the cancel button for the form]	

All other properties of dialogs will be left to their default values unless otherwise stated.

Windows and Dialogs will be invoked by first instantiating an instance of the form and then invoking its Show() or ShowDialog() method. Once the form is no longer needed it's Dispose() method is to be invoked.

```
Dim dlg As New Wic.UI.Common.SelectLocation
dlg.ShowDialog()
dlg.Dispose()
```

The following example illustrates the techniques that are to be used when exchanging data between interfaces. In response to the menu being clicked, we set the dialog's AppointmentId property and update the list item's OnPremisesTime.

```
    Private Sub miMarkApptKept_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles miMarkApptKept.Click
        Me.Cursor = Cursors.WaitCursor
        Dim dlg As New frmMarkApptKept
        Dim ParticipantListItems As ParticipantListItems
        Dim SelectedRowIndex As Integer
        SelectedRowIndex = Me.dgParticipantList.CurrentCell.RowNumber
        ParticipantListItems = Me.dgParticipantList.DataSource
        dlg.ApptId = ParticipantListItems.Item(SelectedRowIndex + 1) ' grid is 0 based
        Me.Cursor = Cursors.Default
        If dlg.ShowDialog() = DialogResult.OK Then
            ' update the on premises time of the participant in the collection
            ParticipantListItems.Item(SelectedRowIndex + 1).OnPremisesTime =
dlg.OnPremisesTime 'grid is 0 based
        End If
        dlg.Dispose()
    End Sub
```

During construction, if an interface invokes another interface that has not been constructed yet the developer must "stub in" and comment out the invocation. It will be the responsibility of the same developer to uncomment this code when the interface becomes available.

12.1.3 Passing Data between Interfaces

Interfaces will occasionally invoke or interact with other interfaces. Calling interface should provide as much relevant data as possible to minimize the round-trip interaction of the interface being called. This should be accomplished by having the interfaces provide properties that can be used to exchange information. Consider the following scenario. While saving a new appointment,

a confirmation dialog is displayed. The information is packaged into an Appointment object which is then passed to the confirmation dialog via it's Appointment property. Upon confirming the Appointment the confirmation dialog then invokes the AddAppointment method of the CalendarServicesAgent. Similarly, when selecting to view a participant's folder the participant list application will invoke the participant folder application passing the ParticipantId as an argument of the invocation.

12.1.4 Controlling the Application Cursor

Most applications at some point perform processing that requires the user to wait for a short period of time. When this occurs the application changes the mouse pointer (also referred to as the cursor) to the Wait Cursor thereby blocking input from the user, once the application has finished the aforementioned processing it changes the cursor back to the Default. Manipulation of the cursor should only be done within an event processing procedure, not within routines that are called by the event processing procedures or within class methods. Cursor control is clearly a user interface paradigm and must remain in the user interface layer as close to the interaction point as possible. Placing cursor control in reusable routines may sound like a good idea but keep in mind you cannot ensure that the routine will not be used in a situation where cursor control in either not appropriate, or is already being performed by the calling procedure.

```
Private Sub btnSearch_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnSearch.Click
    Me.Cursor = System.Windows.Forms.Cursors.WaitCursor
    .
    ' Do some stuff, call some class methods, etc.
    .
    Me.Cursor = System.Windows.Forms.Cursors.Default
End Sub
```

Interfaces should display the wait cursor any time the application must seize control from the user for a noticeable period of time. For example, it would be appropriate the display the wait cursor while retrieving a list. It would not be appropriate to display the wait cursor while setting a member variable in the click event procedure of a checkbox as this operation is not noticeable to the user.

12.1.5 Controlling Application Focus

Most applications will place the focus on the first control in error when some type of business rule violation occurs. This functionality is provided by the default exception handler provided the tag property of the controls point to the business object property that it represents.

```
Me.txtLastName.Tag = "LastName"
```

12.1.6 Populating DataGrid Controls

Displaying data in a datagrid control can be accomplished using helper routines provided by the CVNS.BPDS framework. Below is an example that attaches an instance of the ParticipantListItems collection class to a datagrid control. The example uses the AddColumnToTableStyle shared method of the GuiUtils class to define the column to property mapping of the objects in the collection. (Note that AddCheckboxToTableStyle and AddComboboxToTableStyle are also provided.)

```
Private Sub AddClinicSearchColumnsToGrid()
    Dim t As New System.Windows.Forms.DataGridTableStyle
```

```
    t.RowHeadersVisible = False
    t.MappingName = "ParticipantListItems"
    GuiUtils.AddColumnToTableStyle(t, "HouseholdID", "Household ID", 100)
    GuiUtils.AddColumnToTableStyle(t, "StateWicID", "State WIC ID", 100)
    GuiUtils.AddColumnToTableStyle(t, "LastName", "Last Name", 200)
    GuiUtils.AddColumnToTableStyle(t, "FirstName", "First Name", 100)
    GuiUtils.AddColumnToTableStyle(t, "MiddleInitial", "MI", 20)
    GuiUtils.AddColumnToTableStyle(t, "DateOfBirth", "Date of Birth", 100, "MM/dd/yyyy")
    GuiUtils.AddColumnToTableStyle(t, "WICStatus", "WIC Status", 100)
    Me.dgParticipantList.TableStyles.Add(t)
    Me.dgParticipantList.AllowSorting = True
End Sub
```

Subsequent to setting up the datagrid as depicted above the control is then bound to the collection as follows:

```
Me.dgParticipantList.DataSource = oParticipantListItems
```

All datagrids are to be configured programmatically as depicted above. The IDE designer should not be used to configure a datagrid control.

12.1.7 Providing Sorting and Selection Capability for DataGrid Controls

To enable sorting by columns on a data grid declare 2 member variables that will be store the current sort column and direction.

```
Private _SortDirection As Object
Private _SortColumn As String
```

Next, set the AllowSorting property of the grid to True. For consistency sake this property should be set immediately after the columns are added to the date grid.

```
Private Sub AddClinicSearchColumnsToGrid()
    Dim t As New System.Windows.Forms.DataGridTableStyle
    t.RowHeadersVisible = False
    t.MappingName = "ParticipantListItems"
    GuiUtils.AddColumnToTableStyle(t, "HouseholdID", "Household ID", 100)
    GuiUtils.AddColumnToTableStyle(t, "StateWicID", "State WIC ID", 100)
    GuiUtils.AddColumnToTableStyle(t, "LastName", "Last Name", 200)
    GuiUtils.AddColumnToTableStyle(t, "FirstName", "First Name", 100)
    GuiUtils.AddColumnToTableStyle(t, "MiddleInitial", "MI", 20)
    GuiUtils.AddColumnToTableStyle(t, "DateOfBirth", "Date of Birth", 100, "MM/dd/yyyy")
    GuiUtils.AddColumnToTableStyle(t, "WICStatus", "WIC Status", 100)
    Me.dgParticipantList.TableStyles.Add(t)
    Me.dgParticipantList.AllowSorting = True
End Sub
```

Finally, process the MouseUp event of the datagrid using the shared SortDataGrid() method of the GuiUtils class in CVNS.BPDS framework. Use the HitTestInfo to determine if a column heading or row was selected. Notice that if the column heading was not selected, we default to selecting the row containing the current [selected] cell.

```
Private Sub SelectRow(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles dgParticipantList.MouseUp
    Dim pt = New Point(e.X, e.Y)
    Dim hti As DataGrid.HitTestInfo = sender.HitTest(pt)
    If hti.Type = DataGrid.HitTestType.ColumnHeader Then
        GuiUtils.SortDataGrid(Me.dgParticipantList,

Me.dgParticipantList.TableStyles.Item(0).GridColumnStyles.Item(hti.Column).MappingName,
```

```
                    _SortColumn,
                    _SortDirection)
    Else
        SelectRow()
    End If
End Sub

Private Sub SelectRow()
    dgParticipantList.Select(Me.dgParticipantList.CurrentCell.RowNumber)
    Dim oParticipantListItem As ParticipantListItem
    oParticipantListItem =
CType(Me.dgParticipantList.DataSource.item(Me.dgParticipantList.CurrentCell.RowNumber),
ParticipantListItem)
    With oParticipantListItem
        Me.lblGenderValue.Text = .Gender
        Me.lblWicStatusValue.Text = .WicStatus
        GuiUtils.SetLabelTextAsFormattedDate(Me.lblTerminationDateValue,
.TerminationDate)
        GuiUtils.SetLabelTextAsFormattedDate(Me.lblCertEffValue, .CertStartDate)
        GuiUtils.SetLabelTextAsFormattedDate(Me.lblCertEndValue, .CertEndDate)
    End With
End Sub
```

12.1.8 Other Data Bound Controls

All data bound controls are to be configured programmatically. The IDE designer should not be used to configure a data bound control.

12.1.9 Using Label Controls to Display data elements

Throughout the applications there a several instances where a label control is used to display data in a read-only fashion. When this is the case the foreground color of the label text is to be set to System.Drawing.SystemColors.Highlight.

12.1.10 Third Party Tools

The NetAdvantage Tools from Infragistics will be used to augment the .NET windows forms controls. Several controls have been identified and approved for use within the software.

- The UltraMaskedEdit control is to be used when the interface requires that the values being entered must be formatted in a specific fashion. (i.e Phone numbers, numbers only, restrictive entry)
- The UltraCalendar Combo control is to be used when the interface requires the user to either enter or select a date.
- The UltraDayView control is to be used when displaying daily appointments. Note this control requires the use of UltraCalendarInfo and UltraCalendarLook controls.
- The UltraMonthViewSingle control is to be used when displaying a summary month of appointments. Note this control requires the use of UltraCalendarInfo and UltraCalendarLook controls.
- The UltraCalendarInfo control is to be used as the CalendarInfo source for the UltraDayView and UltraMonthViewSingle controls.
- The UltraCalendarLook control is to be used as the CalendarLook source for the UltraDayView and UltraMonthViewSingle controls.

- The UltraButton control is to be used when a image must be displayed on the button's surface. In order to set the image the button's Appearance.Image property must be set accordingly.

Any additional controls used from this toolkit will require approval from the Spirit Project Application Architect.

12.1.11 Working with the Calendar Controls

As stated above, the calendar controls are used in an interrelated fashion. A number of steps must be performed to establish the proper relationships and populate the controls. Once these are established the interaction between the user and the controls is automatically synchronized.

For an example using the UltraDayView , UltraMonthViewSingle, UltraCalendarInfo, and UltraCalendarLook controls please refer to the form frmScheduleHouseholdAppt in the Appointments project in the SPIRITNAV solution.

12.2 Business Service Layer

The Business Services Layer will be implemented as a number of small libraries based on functional responsibilities, as well as cohesion and collaboration relationships. Within these libraries a mixture of business entity and business workflow classes will be defined.

12.2.1 Entities

Business entity classes are templates used store and manage stateful information. These classes are derived from CVNS.BPDS.DataAccess.BusinessObject and CVNS.BPDS.DataAccess.BusinessObjectCollection and as such inherit all the necessary infrastructure to interact with the a database helper class. Classes derived from BusinessObject represent row level information. Classes derived from BusinessObjectCollection provide strongly typed collections of classes derived from BusinessObject. Most entity classes will map to a table or view within the database (e.g. ParticipantListItem, VendorListItem). Some classes will be provided as a convenience for organizing and manipulating related information (e.g. Person, Height, and Weight). In general, these classes should limit there functionality encapsulated related data and any relational rules associated with that data.

The typical business entity class derived from BusinessObject will aggregate a field map collection by utilizing a shared method of a corresponding mapping class derived from CVNS.BPDS.MappingBase as illustrated below.

```
Public Class ParticipantListItemMapping
   Inherits Base.MappingBase

    Public Shared Function GetMappings() As FieldMappings
        GetMappings = New FieldMappings
        With GetMappings
            .TableName = "V_PARTICIPANT_LIST"
            .Add(New FieldMapping("StateWicID", "STATEWICID", True, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 8))
            .Add(New FieldMapping("HouseholdID", "HOUSEHOLDID", False, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 8))
            .Add(New FieldMapping("LastName", "LASTNAME", False, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 25))
            .Add(New FieldMapping("FirstName", "FIRSTNAME", False, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 20))
```

```
            .Add(New FieldMapping("MiddleInitial", "MIDDLEINITIAL", False, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 1))
            .Add(New FieldMapping("DateOfBirth", "DATEOFBIRTH", False, "Date",
CVNS.BPDS.DataAccess.DataType.DateTime))
            .Add(New FieldMapping("WicStatus", "WICSTATUS", False, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 135))
            .Add(New FieldMapping("Gender", "GENDER", False, "String",
CVNS.BPDS.DataAccess.DataType.VarChar, 20))
            .Add(New FieldMapping("Terminated", "TERMINATED", False, "Boolean",
CVNS.BPDS.DataAccess.DataType.VarChar, 1))
            .Add(New FieldMapping("CertStartDate", "CertStartDate", False, "Date",
CVNS.BPDS.DataAccess.DataType.DateTime))
            .Add(New FieldMapping("CertEndDate", "CERTIFICATIONDUEDATE", False, "Date",
CVNS.BPDS.DataAccess.DataType.DateTime))
            .Add(New FieldMapping("TerminationDate", "TERMINATEDDATE", False, "Date",
CVNS.BPDS.DataAccess.DataType.DateTime))
        End With
    End Function

End Class

Public Class ParticipantListItem
    Inherits BusinessObject
    .
    .
    .
    Public Sub New()
        MyBase.New()
        InitObject()
    End Sub

    Protected Overrides Sub InitObject()
        Me.FieldMappings = Mapping.ParticipantListItemMapping.GetMappings()
        Me.FieldValues.Load(Me.FieldMappings)
        _HouseholdID = New String("")
        _StateWicID = New String("")
        _LastName = New String("")
        _FirstName = New String("")
        _MiddleInitial = New String("")
        _Gender = New String("")
        _WicStatus = New String("")
        _Terminated = False
    End Sub

End Class
```

Notice the FieldMappings and FieldValues properties of the ParticipantListItem class are set in the InitObject routine. This is required of all classes that represent persisted data. For organizational purposes field mappings will be provided via specialized mapping classes.

The corresponding collection class for ParticipantListItem is named ParticipantListItems and is derived from BusinessObjectCollection. This class need only implement the default Item property in order to provide the strongly typed interface.

```
Public Class ParticipantListItems
    Inherits BusinessObjectCollection

    Default Public Shadows Property Item(ByVal index As Integer) As ParticipantListItem
        Get
            Return CType(MyBase.Item(index), ParticipantListItem)
        End Get
        Set(ByVal Value As ParticipantListItem)
            MyBase.Item(index) = Value
        End Set
    End Property

End Class
```

Since the BusinessObjectCollection class is derived from ArrayList and implements a sort method using an internal comparer class, classes derived from this class can be directly attached to databound enabled controls like the DataGrid. For more information pertaining to attaching classes derived from BusinessObjectCollection and DataGrid controls please refer to the User Interface section of the document.

```
Private Sub GetData()
    Me.Cursor = System.Windows.Forms.Cursors.WaitCursor
    Try
        Dim oParticipantListItems As ParticipantListItems
        ClearGrid()
        AddClinicSearchColumnsToGrid()
        oParticipantListItems = _oPSA.Search(Me.txtStateWicID.Text,
Me.txtHouseholdID.Text, Me.txtLastName.Text.Trim.ToUpper, Me.txtFirstName.Text.ToUpper)
        Me.dgParticipantList.DataSource = oParticipantListItems
        If oParticipantListItems.Count = 0 Then
            MsgBox(MSG_NO_MATCHES, MsgBoxStyle.Information, "Participant List")
        End If
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
    Me.Cursor = System.Windows.Forms.Cursors.Default
End Sub
```

12.2.2 Components

12.2.3 Workflows

Business workflow classes are templates used to provide interfaces to complex interactions between business entity objects. These classes are derived from CVNS.BPDS.DataAccess.BusinessManagerBase. An example of a workflow would be the ParticipantSearch class. As illustrated below, this class provides interfaces to perform specific searches (only 1 is included in the example). In turn, these strongly defined search methods call a private generic search which utilizes entity and data access objects to invoke the search and return the result. Notice that the public method `SearchStatewide` performs the high-level edits to ensure the parameters comply with the stated rules for using this method. In this case, the consumer must provide at least one valued parameter in order to invoke the search.

```
    Public Function SearchStatewide(ByVal programid As String, _
                                    ByVal statewicid As String, _
                                    ByVal householdid As String, _
                                    ByVal lastname As String, _
                                    ByVal firstname As String, _
                                    ByVal middleinitial As String, _
                                    ByVal dateofbirth As String, _
                                    ByVal ssn As String, _
                                    ByVal agencyid As String) As ParticipantListItems
        Dim oParticipantListItems As New ParticipantListItems
        Try
            ' ensure we get at least one of the optional arguments
            If householdid.Length = 0 And _
               statewicid.Length = 0 And _
               lastname.Length = 0 And _
               firstname.Length = 0 And _
               middleinitial.Length = 0 And _
               dateofbirth.Length = 0 And _
               ssn.Length = 0 And _
               agencyid.Length = 0 Then
                ' error no search criteria has been provided
                Throw New Exception("No criteria specified. Operation cancelled.")
            End If
```

```
            oParticipantListItems = Search(programid, _
                                           statewicid, _
                                           householdid, _
                                           lastname, _
                                           firstname, _
                                           middleinitial, _
                                           dateofbirth, _
                                           ssn, _
                                           agencyid, _
                                           "", _
                                           "")
        Catch ex As Exception
            Throw ex
        End Try
        Return oParticipantListItems
    End Function
```

The private method `Search` is coded to support 4 strongly defined public search methods. Notice that this method also performs edits to ensure that required criteria of all search types is provided. As stated earlier, this method uses an instance of the SqlServerHelper class and Filters class to return a ParticipantListItems object. (For additional information in regard to the SqlServerHelper and Filters classes please refer to the Data Access Layer section of this document.)

```
    Private Function Search(ByVal programid As String, _
                            ByVal statewicid As String, _
                            ByVal householdid As String, _
                            ByVal lastname As String, _
                            ByVal firstname As String, _
                            ByVal middleinitial As String, _
                            ByVal dateofbirth As String, _
                            ByVal ssn As String, _
                            ByVal agencyid As String, _
                            ByVal clinicid As String, _
                            ByVal apptdate As String) As ParticipantListItems
        Dim oParticipantListItems As New ParticipantListItems
        Try
            If _oDBHelper Is Nothing Then
                Throw (New Exception("DBHelper object is nothing. Operation cancelled."))
            End If
            ' ensure we get the required arguments
            If programid.Length = 0 Then
                Throw New Exception("Program ID is required. Operation cancelled.")
            End If
            ' Add a filter for each of the search criteria.
            Dim oFilters As New Filters
            If programid.Length > 0 Then
                oFilters.Add(New Filter("ProgramID", OperatorType.otEquals, programid))
            End If
            If householdid.Length > 0 Then
                oFilters.Add(New Filter("HouseholdID", OperatorType.otLike, householdid +
"%"))
            End If
            If statewicid.Length > 0 Then
                oFilters.Add(New Filter("StateWicID", OperatorType.otLike, statewicid +
"%"))
            End If
            If lastname.Length > 0 Then
                Dim oChildFilters As New Filters
                ' do a soundex and like or'd together to get all possible matches
                oChildFilters.Add(New Filter("LastName", OperatorType.otSoundsLike,
lastname))
                oChildFilters.Add(New Filter("LastName", OperatorType.otLike, lastname +
"%"), ComparerType.ctOr)
                oFilters.Add(oChildFilters)
            End If
            If firstname.Length > 0 Then
                ' do a soundex and like or'd together to get all possible matches
                Dim oChildFilters As New Filters
```

```
                oChildFilters.Add(New Filter("FirstName", OperatorType.otSoundsLike,
firstname))
                oChildFilters.Add(New Filter("FirstName", OperatorType.otLike, firstname
+ "%"), ComparerType.ctOr)
                oFilters.Add(oChildFilters)
            End If
            If middleinitial.Length > 0 Then
                oFilters.Add(New Filter("MiddleInitial", OperatorType.otEquals,
middleinitial))
            End If
            If dateofbirth.Length > 0 Then
                If IsDate(dateofbirth) Then
                    oFilters.Add(New Filter("DateOfBirth", OperatorType.otEquals,
(CType(dateofbirth, Date))))
                Else
                    ' error no search criteria has been provided
                    Throw New Exception("Invalid date of birth date format. Operation
cancelled.")
                End If
            End If
            If agencyid.Length > 0 Then
                oFilters.Add(New Filter("AgencyID", OperatorType.otEquals, agencyid))
            End If
            If clinicid.Length > 0 Then
                oFilters.Add(New Filter("ClinicID", OperatorType.otEquals, clinicid))
            End If
            If apptdate.Length > 0 Then
                If IsDate(apptdate) Then
                    oFilters.Add(New Filter("ApptDate", OperatorType.otEquals,
(CType(apptdate, Date))))
                Else
                    ' error no search criteria has been provided
                    Throw New Exception("Invalid appointment date format. Operation
cancelled.")
                End If
            End If
            ' instantiate the target collection and invoke the search.
            _oDBHelper.Search(Mapping.ParticipantListItemMapping.GetMappings,
oParticipantListItems, oFilters)
        Catch ex As Exception
            Throw ex
        End Try
        Return oParticipantListItems
    End Function
```

12.3 Web Service Interface Layer

Web service interfaces will be implementation and platform agnostic. Method arguments will be provided as base data types (string, date, integer, etc.) as defined in the W3C specification. Note that xml inputs and outputs xml will be typed as strings. This design pattern allows for future reuse of these services by a variety of consumer applications. Note that since the aforementioned xml is only used by the underlying framework to stream serialized objects between the consumer and the service there is no need at this time to a schema definition. Applications and components within this solution will be provided with a framework that will provide tools to translate XML to and from solution specific class types. Under no circumstances should any component construct xml "on-the-fly". Also, all web services return data as strings. These returned string must be processed with the translator method ReadXMLResponse.

Web services will provide interface wrappers around business workflows and entities. In the example provided below the `SearchStatewide` method of the ParticipantServices web service provides a strongly defined search interface. In fact, this search is very similar to the `SearchStatwide` interface provided by the workflow class ParticipantSearch. Upon

examination of this web method we can see that it also uses a generic private `Search` routine that in turn uses the aforementioned ParticipantSearch workflow to invoke the actual search.

Notice that an instance of the SqlServerHelper class is created here and passed to the workflow object. This is done because the application (in this case the web service) is responsible for knowing where the data store is located. The workflow object returns an instance of the ParticipantListItems collection class. The ParticipantListItems collection is then serialized to xml and wrapped in a response envelope by an instance of the Translator class. (More information on SQLServerHelper, Filters, and the Translator class can be found in the *Data Access Layer* section of this document.)

```
<WebMethod()> _
Public Function SearchStatewide(ByVal programid As String, _
                                ByVal statewicid As String, _
                                ByVal householdid As String, _
                                ByVal lastname As String, _
                                ByVal firstname As String, _
                                ByVal middleinitial As String, _
                                ByVal dateofbirth As String, _
                                ByVal ssn As String, _
                                ByVal agencyid As String) As String
    Return Search(ParticipantSearchTypeEnum.pstStatewide, _
                  programid, _
                  statewicid, _
                  householdid, _
                  lastname, _
                  firstname, _
                  middleinitial, _
                  dateofbirth, _
                  ssn, _
                  agencyid, _
                  "", _
                  "")
End Function

Private Function Search(ByVal st As ParticipantSearchTypeEnum, _
                        ByVal programid As String, _
                        ByVal statewicid As String, _
                        ByVal householdid As String, _
                        ByVal lastname As String, _
                        ByVal firstname As String, _
                        ByVal middleinitial As String, _
                        ByVal dateofbirth As String, _
                        ByVal ssn As String, _
                        ByVal agencyid As String, _
                        ByVal clinicid As String, _
                        ByVal apptdate As String) As String
    Dim EndDttm As New DateTime
    Dim StartDttm As New DateTime
    Dim s As New String("")
    Dim oTranslator As New Translator
    Dim obj As Object
    StartDttm = Now
    Try
        ' intantiate a SqlServerHelper that we can use to interact with the database.
        Dim oParticipantSearch As New ParticipantSearch
        Dim oParticipantListItems As New ParticipantListItems
        With oParticipantSearch
            ' hand off the helper to the object and invoke the search
            If st = ParticipantSearchTypeEnum.pstStatewide Then
                ' Limit the result set to no more than 100 rows.
                oParticipantListItems = .SearchStatewide(programid, _
                                                         statewicid, _
                                                         householdid, _
                                                         lastname, _
                                                         firstname, _
                                                         middleinitial, _
```

```
                                                    dateofbirth, _
                                                    ssn, _
                                                    agencyid)
            ElseIf st = ParticipantSearchTypeEnum.pstClinic Then
                ' Limit the result set to no more than 100 rows.
                oParticipantListItems = .SearchClinic(clinicid, _
                                                      programid, _
                                                      statewicid, _
                                                      householdid, _
                                                      lastname, _
                                                      firstname, _
                                                      middleinitial, _
                                                      dateofbirth, _
                                                      ssn)
            ElseIf st = ParticipantSearchTypeEnum.pstOnSite Then
                ' no limit no the result set.
                oParticipantListItems = .SearchOnSite(clinicid, _
                                                      programid, _
                                                      statewicid, _
                                                      householdid, _
                                                      lastname, _
                                                      firstname, _
                                                      middleinitial, _
                                                      dateofbirth, _
                                                      ssn)
            ElseIf st = ParticipantSearchTypeEnum.pstDailyAppointments Then
                ' no limit no the result set.
                oParticipantListItems = .SearchDailyAppointments(clinicid, _
                                                                 apptdate, _
                                                                 programid, _
                                                                 statewicid, _
                                                                 householdid, _
                                                                 lastname, _
                                                                 firstname, _
                                                                 middleinitial, _
                                                                 dateofbirth, _
                                                                 ssn)
            End If
        End With
        oParticipantSearch = Nothing
        obj = oParticipantListItems
    Catch ex As Exception
        obj = ex
    Finally
        EndDttm = Now
        Dim ts As New TimeSpan(EndDttm.Ticks - StartDttm.Ticks)
        ' translate the object to xml and wrap it in a response envelope
        oTranslator.WriteXMLResponse(obj, _
                                     s, _
                                     BuildTransactionId(Me.User.Identity.Name), _
                                     ts.Milliseconds.ToString)
        oTranslator = Nothing
    End Try
    Return s
End Function
```

12.4 Web Service Agent Layer

Some web services will have corresponding client side agents. These agents wrap client-side web service proxy interface stubs providing the ability to route requests to either the remote web service or the local database (via the workflow objects) based on the connectivity state of the application. This capability is being provided as part of the "smart client technology" project initiative. Classes that act as client-side agents will implement the IServiceAgent interface in order to ensure that the agent provides a property that consumers can use to indicate the current online/offline state of the application.

The following example from the ParticipantServicesAgent class illustrates the use of a client side agent. Notice the use of the class' OnLine property to determine the request routing. When on line the request is routed to the web service. Offline, the request is serviced locally using code that is very similar to that of the corresponding web service.

```
Public Function SearchStatewide(ByVal programid As String, _
                                ByVal statewicid As String, _
                                ByVal householdid As String, _
                                ByVal lastname As String, _
                                ByVal firstname As String, _
                                ByVal middleinitial As String, _
                                ByVal dateofbirth As String, _
                                ByVal ssn As String, _
                                ByVal agencyid As String) As ParticipantListItems
    Return Search(ParticipantSearchTypeEnum.pstStatewide, _
                  programid, _
                  statewicid, _
                  householdid, _
                  lastname, _
                  firstname, _
                  middleinitial, _
                  dateofbirth, _
                  ssn, _
                  agencyid, _
                  "", _
                  "")
End Function

Private Function Search(ByVal st As ParticipantSearchTypeEnum, _
                        ByVal programid As String, _
                        ByVal statewicid As String, _
                        ByVal householdid As String, _
                        ByVal lastname As String, _
                        ByVal firstname As String, _
                        ByVal middleinitial As String, _
                        ByVal dateofbirth As String, _
                        ByVal ssn As String, _
                        ByVal agencyid As String, _
                        ByVal clinicid As String, _
                        ByVal apptdate As String) As ParticipantListItems
    Dim oParticipantListItems As New WICLIB.ParticipantListItems
    Dim sTransactionid As String
    Dim sTimespan As String
    If OnLine Then
        Try
            ' get the data from the web service
            Dim s As String
            If st = ParticipantSearchTypeEnum.pstStatewide Then
                s = _oPS.SearchStatewide(True, _
                                         programid, _
                                         statewicid, _
                                         householdid, _
                                         lastname, _
                                         firstname, _
                                         middleinitial, _
                                         dateofbirth, _
                                         ssn, _
                                         agencyid)
            ElseIf st = ParticipantSearchTypeEnum.pstClinic Then
                s = _oPS.SearchClinic(True, _
                                      clinicid, _
                                      programid, _
                                      statewicid, _
                                      householdid, _
                                      lastname, _
                                      firstname, _
                                      middleinitial, _
                                      dateofbirth, _
                                      ssn)
```

```
            ElseIf st = ParticipantSearchTypeEnum.pstOnSite Then
                s = _oPS.SearchOnSite(True, _
                                      clinicid, _
                                      programid, _
                                      statewicid, _
                                      householdid, _
                                      lastname, _
                                      firstname, _
                                      middleinitial, _
                                      dateofbirth, _
                                      ssn)
            ElseIf st = ParticipantSearchTypeEnum.pstDailyAppointments Then
                s = _oPS.SearchDailyAppointments(True, _
                                                 clinicid, _
                                                 apptdate, _
                                                 programid, _
                                                 statewicid, _
                                                 householdid, _
                                                 lastname, _
                                                 firstname, _
                                                 middleinitial, _
                                                 dateofbirth, _
                                                 ssn)
            End If
            Dim oTranslator As New Translator
            oTranslator.ReadXMLResponse(s, _
                                        oParticipantListItems, _
                                        sTransactionid, _
                                        sTimespan)
            oTranslator = Nothing
        Catch ex As Exception
            Throw ex
        End Try
    Else
        ' get the data from the local data store
        Try
            Dim oParticipantSearch As New ParticipantSearch
            With oParticipantSearch
                If st = ParticipantSearchTypeEnum.pstStatewide Then
                    oParticipantListItems = .SearchStatewide(programid, _
                                                             statewicid, _
                                                             householdid, _
                                                             lastname, _
                                                             firstname, _
                                                             middleinitial, _
                                                             dateofbirth, _
                                                             ssn, _
                                                             agencyid)
                ElseIf st = ParticipantSearchTypeEnum.pstClinic Then
                    oParticipantListItems = .SearchClinic(clinicid, _
                                                          programid, _
                                                          statewicid, _
                                                          householdid, _
                                                          lastname, _
                                                          firstname, _
                                                          middleinitial, _
                                                          dateofbirth, _
                                                          ssn)
                ElseIf st = ParticipantSearchTypeEnum.pstOnSite Then
                    oParticipantListItems = .SearchOnSite(clinicid, _
                                                          programid, _
                                                          statewicid, _
                                                          householdid, _
                                                          lastname, _
                                                          firstname, _
                                                          middleinitial, _
                                                          dateofbirth, _
                                                          ssn)
                ElseIf st = ParticipantSearchTypeEnum.pstDailyAppointments Then
                    oParticipantListItems = .SearchDailyAppointments(clinicid, _
                                                                     apptdate, _
```

```
                                                programid, _
                                                statewicid, _
                                                householdid, _
                                                lastname, _
                                                firstname, _
                                                middleinitial, _
                                                dateofbirth, _
                                                ssn)
                End If
            End With
        Catch ex As Exception
            Throw ex
        End Try
    End If
    Return oParticipantListItems
End Function
```

In either case, the agent returns an instance of the ParticipantListItems collection.

12.5 Data Access Layer

A runtime trust boundary is preserved between the Business Service Layer and the Data Access Layer by using objects instantiated from the CVNS.BDPS.DataAccess Framework. By passing the source and or target objects derived from CVNS.BPDS.BusinessObject or CVNS.BPDS.BusinessObjectCollection as method arguments.

These runtime objects provide an abstracted object model to the data store allowing the developer to work at the object level and avoid writing SQL associated with the basic CRUD (Create, Read, Update, and Delete) operations. This object model provides a framework that supports the majority of SQL queries that are typically written. (Note that the SqlServerHelper class derived from DBHelper also provides a direct interface for executing [complex] SQL statements that are beyond the capabilities of the current the object model.

12.5.1 SqlServerHelper and Filters

The Data Access Layer provides a framework of helper classes that can be used to interact with the appropriate persistence mechanism (e.g. databases, files, etc.) The following illustrates using the framework to retrieve a collection of ParticipantListItem objects where the participant's last name starts with "NA" and first name starts with "B". Notice the mapping information of the objects in the collection, the target collection, and the filter criteria are passed in as arguments of the method invocation.

```
Dim oFilters As New Filters
oFilters.Add(New Filter("LastName", OperatorType.otLike, "NA%"))
oFilters.Add(New Filter("FirstName", OperatorType.otLike, "B%"))
Dim oList As New ParticipantListItems
oDBHelper.Search(Mapping.ParticipantListItemMapping.GetMappings, oList, oFilters)
Return oList
```

The Filters and Filter classes provide an object oriented interface that can be used to define the criteria for the search. The Add method of the Filters collection class provides an optional argument that can be used to define the and/or relationship of the filter being added. For example, the following code would generate criteria to locate participants with a WIC status of *infant* or *child*. (The default comparer is `ComparerType.ctAnd`)

```
Dim oFilters As New Filters
oFilters.Add(New Filter("WicStatus", OperatorType.otEquals, "I"))
oFilters.Add(New Filter("WicStatus", OperatorType.otEquals, "C"), ComparerType.ctOr)
```

```
Dim oList As New ParticipantListItems
oDBHelper.Search(Mapping.ParticipantListItemMapping.GetMappings, oList, oFilters)
Return oList
```

The Filters class supports a heterogeneous collection of Filter and Filters objects. This allows the consumer to construct nested criteria. For example to locate all participants whose last name sounds like 'pat' or starts with 'pat' and has a wic status of 'child' we would construct the following Filters collection.

```
Dim oChildFilters As New Filters
' do a soundex and like or'd together to get all possible matches
oChildFilters.Add(New Filter("LastName", OperatorType.otSoundsLike, lastname))
oChildFilters.Add(New Filter("LastName", OperatorType.otLike, lastname + "%"),
ComparerType.ctOr)
oFilters.Add(oChildFilters)
oFilters.Add(New Filter("WicStatus", OperatorType.otEquals, "C"))
```

Invoking the `toString` method of the this Filters object would return the following string:

(((soundex([lastname]) = soundex('PAT') or ([lastname]) like 'PAT%') and ([wicstatus] = 'C'))

Other basic CRUD (Create, Read, Update, Delete) operations are supported as well. For example to retrieve, access, and save a specific item we would write the following code:

```
Dim oItem As New ParticipantListItem
Dim s As New String
' get the data
oItem.StateWicID = "00807111"
oDBHelper.Load(oItem)
' make changes
s = oItem.LastName
.
.
oItem.LastName = s
' save changes
oDBHelper.Save(oItem)
```

Similarly, to delete the item we would use the following code.

```
.
.
.
oDBHelper.Delete(oItem)
```

12.5.2 Translator

The translator class implements methods that provide for the transformation of objects derived from BusinessObject and BusinessObjectCollection to and from xml. These transformation services rely on the FieldMappings contained in each object. Consider the following method from the ParticipantServicesAgent class that uses an instatance of the Translator class to serialize the ParticipantDemographics object in order to submit the information to the corresponding web service.

```
Public Function SaveDemographics(ByVal pd As ParticipantDemographics) As String
    Dim oTranslator As New Translator
    Dim xml As String
    Dim s As String
    oTranslator.toXML(pd, xml)
    s = _oPS.SaveDemographics(xml)
    Return s
End Function
```

Likewise, the corresponding web service uses the fromXML method of the translator to deserialize the xml back into a ParticipantDemographics object which is then used as an argument of the Save method of the SqlServerHelper object.

```
Dim oPD As New ParticipantDemographics
oTranslator.fromXML(xml, oPD)
Dim oDBHelper As SQLServerHelper
oDBHelper = GetDBHelper(Me.User.Identity.Name)
oDBHelper.Save(oPD)
```

The Translator class also implements the WriteXMLResponse method which provides a wrapper over the toXML method and is used by web service methods to format the response accordingly.

```
oTranslator.WriteXMLResponse(obj, s, BuildTransactionId(Me.User.Identity.Name),
ts.Milliseconds.ToString)
```

12.5.3 BusinessObject

The BusinessObject base class implements the infrastructure required to interact with the data access helper classes provided within this framework. Classes derived from this class must aggregate *FieldMappings* which provide the property to column mappings between the class and the associate database table or view.

12.5.4 Field Mappings

Filed mapping classes contain class property to database column mappings.

12.5.5 BusinessObjectCollection

The BusinessObjectCollection class is derived from the .NET framework's ArrayList class and provides a base class from which to derive strongly typed collections of classes derived from BusinessObject.

12.6 XP Interface Compatibility

Each application that wants to use the Windows XP control set must have the control set readily available and must use a manifest file to activate the set. The file must be named [MyApp].exe.manifest where [MyApp] the name of the application's executable. The manifest file must be in the same folder as the executable. Below is the corresponding manifest file for ParticipantList.exe.

```
<?xml version="1.0" encoding="utf-8" ?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="X86"
    name="ParticipantList"
    type="win32"
/>
<description>ParticipantList</description>
<dependency>
    <dependentAssembly>
        <assemblyIdentity
            type="win32"
            name="Microsoft.Windows.Common-Controls"
            version="6.0.0.0"
            processorArchitecture="X86"
            publicKeyToken="6595b64144ccf1df"
            language="*"
        />
```

```
    </dependentAssembly>
</dependency>
</assembly>
```

Note that bolded information must correspond with the associated application. In addition, controls that support the "FlatStyle" property must be set to "System"

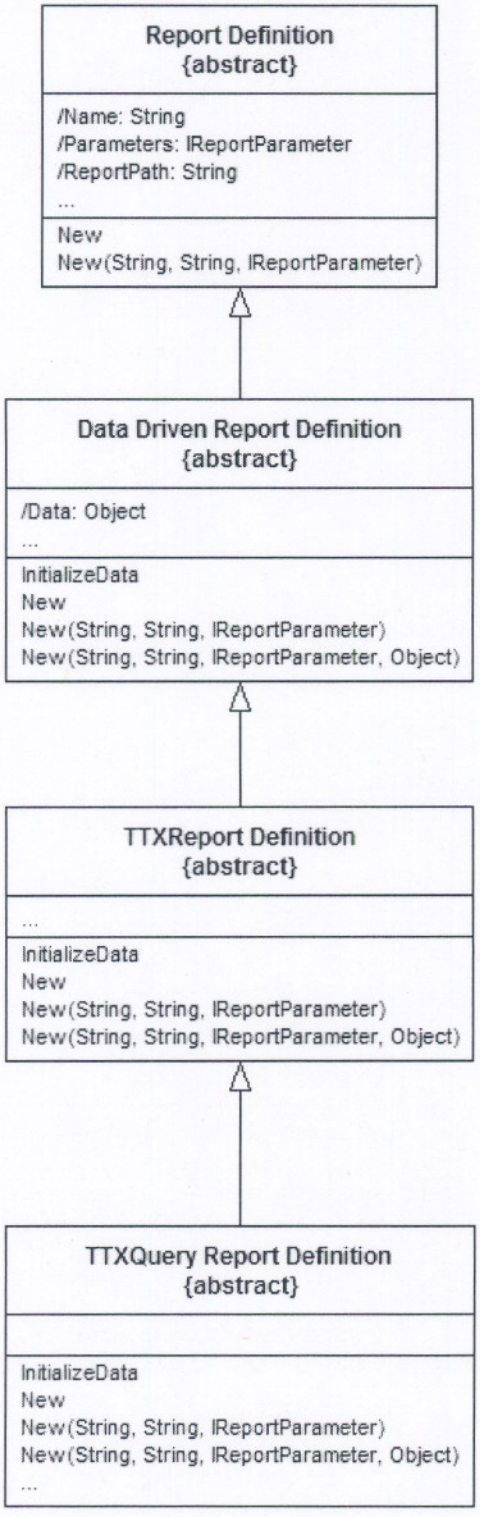
12.7 Reporting Component Architecture

A brief overview of the architecture behind the reporting system is needed to effectively convert existing reports and build new reports.

12.7.1 Report Definitions

Report definitions are classes that define the report to be used in the application. They include the name of the report, the report parameters, and the path to the report's RPT file (the Crystal Reports report layout file – .rpt file extension).

Other types of report definitions derived from this base definition include a generic data-driven report definition and more specific report definitions for reports using TTX files. A class hierarchy of these classes is shown below.

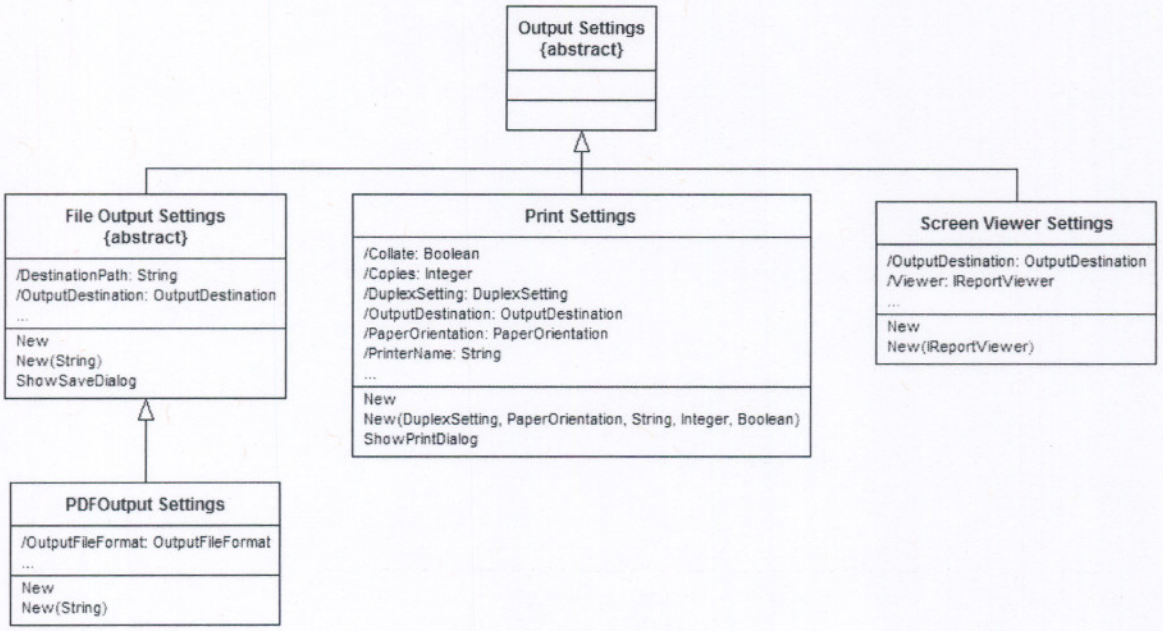


All provided report definition classes are abstract, and each report used in the system should have its own specifically implemented report definition class that inherits from one of these system provided classes.

12.7.2 Output Settings

Output settings are classes that define where the output of the report should go. Currently, three output settings are provided by the reporting component: Print, Screen Viewer, and PDF Output file. Each type of setting has specific properties that must be set to output the report properly.

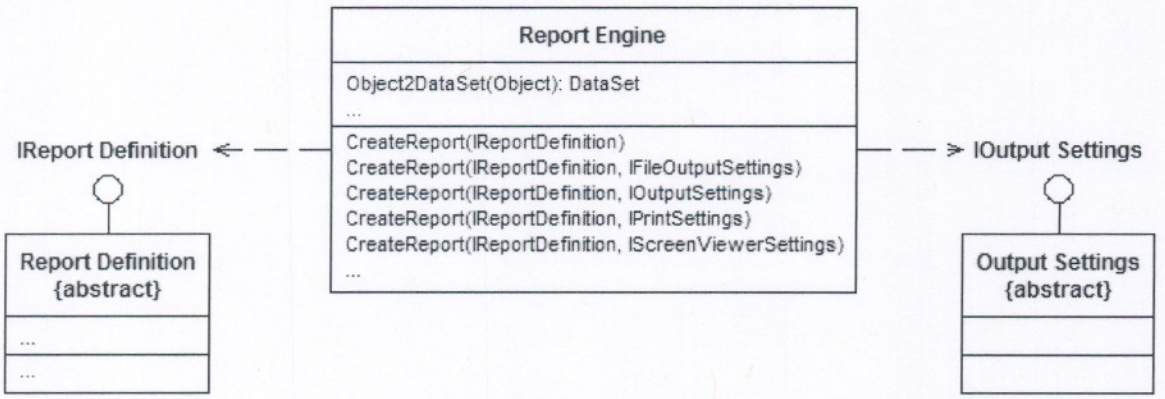
Here is a class diagram showing the class hierarchy of the various output settings:



12.7.3 Report Engine

The ReportEngine class does all of the work of generating the report. At a minimum, it needs a report definition specifying what report to create. In this case, a default output setting will be provided. Otherwise, it needs the defined output setting for where the report should be created.

Below is a diagram showing the ReportEngine class's public members and their relationship with ReportDefinition and OutputSettings classes.



12.7.4 Walkthroughs

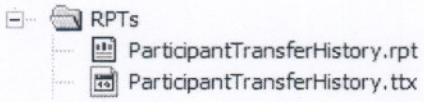
This section goes step-by-step through the process of building the supporting report component classes that will allow your application to quickly and easily begin using the component. The examples in this section will extend from the beginning of the process to the end and will include

specific code and screen shots. The Reports.Prototype solution has a project called ReportingExample that includes working copies of these examples.

12.7.5 Design Reports

The first step in using the reporting component is to design the report. This includes creating a TTX file to layout your report with and building the report in the Crystal Reports report designer. If you are converting a report that has been designed using a TTX file, this step has already been done for you. If the report you are converting connects directly to the database, treat this report as a new report and build both the TTX and RPT files.

After designing the report, add the RPT and TTX files into a Visual Studio project folder. The files should be placed in the same folder. Part of a screen shot below shows an RPT and TTX file in the same folder.



12.7.6 Create Report Definitions

After the report has been designed, you must decide whether the application will already have the data used in the report or not. The type of report definition depends upon this determination and will be discussed momentarily.

12.7.6.1 Properties

Regardless of whether the application provides the data or not, your report definition class will have the following properties: Name, ReportPath, Parameters, Data, and TTXTableMapping. (Even though the data may not be provided by the application, the data will be retrieved before the report is processed, which happens when the report definition and output settings are passed to the ReportEngine class's CreateReport method.)

12.7.6.1.1 Name

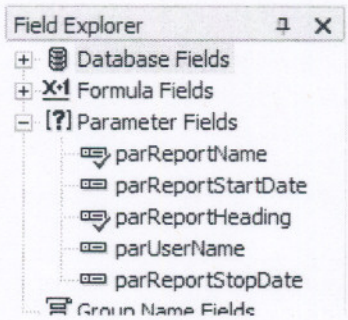
The Name is simply the name of the report. This property is used primarily by the default screen viewer output. It may also be used in custom screen viewers or in derived processing.

12.7.6.1.2 ReportPath

The ReportPath is the full path to the RPT file. Because you should never hard-code a full path, use a dynamic technique for obtaining the full path, such as using the System.AppDomain.CurrentDomain.BaseDirectory path along with a relative path to the RPT file. Using only a relative path will cause problems when outputting a report to the file system.

12.7.6.1.3 Parameters

Parameters for the report may be constants defined within the report definition or variables passed to the report definition via the constructor or set explicitly, but they must be a part of the report definition before the report can be processed. Report parameters are name, value pairs where the name of the parameter must match the parameter name the report is expecting to receive. If the names do not match, the report will assume the parameter was not given. (The report parameters can be found in the Field Explorer, which only displays when viewing the RPT file and displaying the Document Outline window. You can find this window in the menu by going to View | Other Windows | Document Outline.) Here is what parameters look like in the Field Explorer:



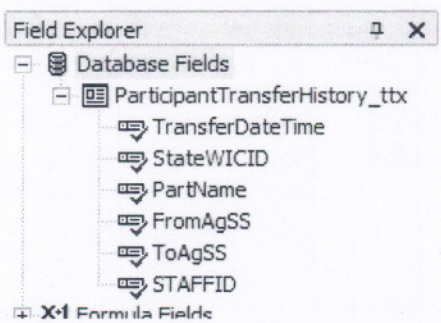
12.7.6.1.4 Data

Report Data is the data that the report will display. Most of the time, when an application will provide the data, the data will be in the form of a collection, and when the data is retrieved from the database, it will be in the form of a DataSet.

12.7.6.1.5 TTXTableMapping

The TTXTableMapping property refers to a statically created class that derives from the abstract TTXTableMapping class. This table mapping class maps the name the report uses for the table to the class name for the data being passed to the report. (This is assuming that the application has the data to be used in the report. If the application does not have this data already and needs to get the data from a data source, the class maps the report's expected table name with the DataSet's table name.)

The table mapping class also has a set of field mapping classes that map the fields the report expects to receive with the properties of the class you pass the report as data. (Again, this assumes the application has the data to be used in the report. If the application does not have this data, this field mapping class maps the expected field names to DataTable column names.) The table and field names the report expects to see can be found in the Crystal Reports Field Explorer, as shown below.



This is what a completed, derived table mapping class looks like:

```
Public Class ParticipantTransferHistoryReportMapping
    Inherits TTXTableMapping

    Public Sub New()
        MyBase.New("ParticipantTransferHistory_ttx", _
"ParticipantTransferHistory")
        Dim FieldMappings(5) As TTXFieldMapping
        FieldMappings(0) = New TTXFieldMapping( _
"TransferDateTime", "TransferDateTime")
        FieldMappings(1) = New TTXFieldMapping( _
"StateWICID", "StateWICID")
        FieldMappings(2) = New TTXFieldMapping( _
"PartName", "ParticipantName")
        FieldMappings(3) = New TTXFieldMapping( _
"FromAgSS", "FromAgency")
        FieldMappings(4) = New TTXFieldMapping("ToAgSS", _
"ToAgency")
        FieldMappings(5) = New TTXFieldMapping("STAFFID", _
"StaffID")
        Me.TTXFieldMappings = FieldMappings
    End Sub

End Class
```

12.7.6.2 Data Provided

If the application has the data to be used in the report, you will need to create a report definition class that inherits from the TTXReportDefinition class. This class has the base implementation to initialize the given report data based on the given mapping class so the report can readily use it.

Here is an example of a data-provided report definition class:

```
Public Class ParticipantTransferHistoryReportDefintion
    Inherits TTXReportDefinition

    ' because of the way Crystal handles paths when saving
files to disk, the full path to the rpt file must be
specified.
    Private Shared ReadOnly ReportPathConst As String = _
String.Concat( _
System.AppDomain.CurrentDomain.BaseDirectory, _
"..\Reports\RPTs\ParticipantTransferHistory.rpt")

    Private Shared ReadOnly ParametersConst As _
IReportParameter() = _
        {New ReportParameter("parReportName", _
"My Test Participant Transfer History Report"), _
        New ReportParameter("parReportStartDate", _
DateTime.Now.Subtract(New TimeSpan(1, 0, 0, 0))), _
        New ReportParameter("parReportHeading", _
"Test Report Heading"), _
        New ReportParameter("parUserName", "Jane Doe"), _
        New ReportParameter("parReportStopDate", _
DateTime.Now)}

    Public Sub New()
        MyBase.New( _
"Participant Transfer History Report", ReportPathConst, _
ParametersConst, New ParticipantTransferHistoryItems)
        MyBase.TTXTableMapping = New _
ParticipantTransferHistoryReportMapping
    End Sub

End Class
```

12.7.6.3 Data Not Provided

On the other hand, if the application does not have the report data readily available, you will need to create a report definition that inherits from the TTXQueryReportDefinition class. This abstract class inherits from the TTXReportDefinition class and overwrites some of its implementation in order to retrieve data from the database as a part of the data initialization.

Classes derived from the TTXQueryReportDefinition class must implement a method called GetQuery. It is in this method where the query to retrieve the data for the report will be defined. Below is an example of a query that is built in the GetQuery method.

```
    Protected Overrides Function GetQuery() As String
        Dim Query As New StringBuilder
        Query.Append("select * ")
        Query.Append("from participanttransferhistory ")
        Query.Append(String.Format( _
"where participantid = '{0}'", Me.m_ParticipantID))
        Return Query.ToString()
    End Function
```

When implementing the GetQuery method, be sure to use the StringBuilder class and the String object's Format method, as shown above, rather than implicit string concatenation, as shown below.

```
    Protected Overrides Function GetQuery() As String
        Dim Query As String
        Query = "select * "
        Query = Query & "from ParticipantTransferHistory "
        Query = Query & "where ParticipantID = " & _
participantID
        Return Query
    End Function
```

12.7.7 Incorporating Reports Into Applications

Assuming that all the report definitions and mappings have been created for the reports you need to use in your application, the process of creating reports is very simple. First, you'll create an instance of your report definition. Second, you'll create an instance of output settings to use when creating your report. Finally, you'll tell the report engine to create the report for you. Here is an example of what the code would look like:

```
Dim Definition As _
      New ParticipantTransferHistoryReportDefinition
Dim Settings As New PDFOutputSettings("C:\report.pdf")
Dim Engine As New ReportEngine
Engine.CreateReport(Definition, Settings)
```

12.7.8 Special Notes

In working with the reporting component, there are some special notes that you must keep in mind.

12.7.8.1 Data Access Component

At the time of this writing, there was not a standard data access component. Thus, the TTXQueryReportDefinition class's base implementation of the InitializeData method is not fully implemented. This component is expected to be completed soon and integrated into the reporting component to complete the described functionaltiy.

12.7.8.2 Temporary Database Tables

Some reports that are to be converted have wrapper classes around them that insert data into a temporary database table only to immediately retrieve it out again for the purpose of passing the data to the report. (As an example, the ParticipantTransferHistory report was done this way.) An implementation like this is no longer needed and should be avoided for the techniques described in this document.

12.8 Dictionary Lists

Dictionary lists are static sets of domain values that can be used to populate drop-down lists and perform code/descriptor translations. These lists are acquired using the shared GetList method of the Dictionary class. The method returns an instance of a DictionaryListItems collection class. This class can in turn be used as a data source for databound controls or for translation purposes programmatically.

```
Dim oDictionaryListItems As New DictionaryListItems
oDictionaryListItems = Dictionary.GetList(Dictionary.DictionaryCategoryEnum.dcProgram,
oDBHelper)
With Me.cboProgram
    .DataSource = oDictionaryListItems
    .ValueMember = "ExternalID"
    .DisplayMember = "Description"
End With
```

Dictionary lists will be updated as required. The dictionary will be stored locally on the client machine by serializing the dataset retrieved from the central data store. Each time the first WIC application is started on the desktop the system will check for the most recent copy of the dictionary, if this copy is newer than the local copy the data sych client will retrieve the most recent dictionary and serialize it to the local hard disk via XML. This list loaded into memory at the start of each application session.

12.9 Growth Grids and CDC Data

The growth grid functionality allows the user to view and compare measurement data against percentile data provided by the CDC. The CDC data has been imported into the database and is stored in the following tables. The original CDC data is provided in metric form (kilograms and centimeters). During the import process a LENGTHWEIGHTAGE summary table was populated using values from the CDC tables. While populating the summary table the metric values in the CDC data were converted to imperial values (kilograms to inches, centimeters to pounds)

The LENGTHWEIGHTAGE table contains 17 chart types; one for each gender/chart type combination (except the BMI and premi charts which use the BMIAGE and PREMATURELENGTHAGE and PREMATUREWEIGHTAGE tables. Note that the premi data did not come from the CDC. The list of types can be found in the LENGTHWEIGHTAGETYPE table.

The data in the LENGTHWEIGHTAGE is used to populate the CDC percentile bands in each grid.

For example, if you where trying to locate the percentile banding for a Boy between 0 to 36 months comparing Length to Age you would use TYPE =1 (BB36LengthAge) in the LENGTHWEIGHTAGE. Each row would give you the length measurement for each percentile so for TYPE = 1 and AGEMONTHS = 1 (a 1 month old boy) a measurement of 19.98032 inches would place the infant in the 5th percentile. The remaining columns on that row designate each of the percentile measurements for that type at that age.

The measurement data percentile is calculated and plotted on the fly. These calculations also vary based on the comparison being performed and are not as easy to describe. Continuing the example from above to calculate the percentage for a length to age comparison for a 1 month old boy you would have to convert the length measurement to centimeters, lookup the appropriate range of records in the LGTHAGE table (in this case SEX = 1 and AGEMOS = 0.5 and 1.5, perform interpolation between the 2 records on the L, M, and S values (since we did not find an exact matching record – our age is 1month, the table has ages 0.5 and 1.5 months). So in the formula below the 0.5 record would be the "low" values and the 1.5 record would be the "high" values and Var is L, M, and S on the corresponding record. These L,M, and S values are then used to calculate a Z score which in turn is converted into a percentage using the PfromZ function in the Dist.dll com library provided by the CDC.

A class has been created that provides a simplified interface to these complex calculations. The Wic.Charting.CdcStatisticsCalculator class will return a comprehensive set of statistical information for each measurement provided.

12.10 Consignment of System-wide Identifiers

The system maintains blocks of identifiers that can be used by specified machines while inserting data into the database. Using these blocks ensures that machine operating in offline mode insert records into their local database using system wide unique values. For example, the StateWicIdConsignment table has a row for each machine that has been allocated a block if ids:

MachineId	StartingId	LastUsedId	EndingId
WebService	1	300	10000

In the example above the WebService machine has been allocated a block of 10,000 numbers starting with 1 and ending with 10,000. To date 250 numbers have been used (mostl likely while client applications were adding new applicants while operating in online mode)

Assume we have a laptop that is capable of operating offline and that his laptop checks out a clinic. During the checkout process a new row is added to the table:

MachineId	StartingId	LastUsedId	EndingId
WebService	1	300	10000
Laptop01	**10001**	**0**	**10250**

Notice that the block assigned to the machine (note that in production actual id would be the laptops MAC Address) has been assigned a block of 250 numbers that do not overlap with the block allocated to the WebService. While offline the laptop is used to add 10 new members. Upon checking in the clinic the table is updated as follows:

MachineId	StartingId	LastUsedId	EndingId
WebService	1	300	10000
Laptop01	10001	**10**	10250

The system has a minimum threshold of numbers that must be in the open block. If the block falls below that minimum threshold the system will allocate another block to ensure that the machine has a sufficient number of ids. Let's assume that the laptop has checked data out several times and the current table data is as follows:

MachineId	StartingId	LastUsedId	EndingId
WebService	1	5500	10000
Laptop01	10001	10160	10250

Assuming that the minimum threshold for the laptop is 100. The next time the laptop checks out a clinic the system will allocate a second block for the laptop as follows:

MachineId	StartingId	LastUsedId	EndingId
WebService	1	5500	10000
Laptop01	10001	10160	10250

Laptop02	10251	30	10500
Laptop01	**10501**	**0**	**10750**

While offline the laptop will use the numbers in the current block first. At the point at which the number 10250 is used, the laptop will switch the second block of numbers. Let's assume that the laptop, while offline has added 100 new members. Upon checking in the clinic the table data would be updated as follows:

MachineId	StartingId	LastUsedId	EndingId
WebService	1	5500	10000
Laptop01	10001	**10250**	10250
Laptop02	10251	30	10500
Laptop01	10501	**10510**	10750

How does it work? The software uses mapping classes to "map" the properties of a class to the columns of a table. In that same mapping class a ValueGenerator is used to indicate the name of the stored procedure to be used to retrieve the next id. Below is a code snippet from the MemberMapping class. Notice the use of the ValueGenerator which references the GetNextWicId stored procedure and returns the Statewicid output value. Also notice that the last argument of the StateWicId property map refers to the ValueGenerator.

```
Public Class MemberMapping
  Inherits MappingBase

Public Shared Function GetMappings() As FieldMappings

   GetMappings = New FieldMappings

   Dim StatewicIDValue As New CVNS.BPDS.ValueGenerator
   StatewicIDValue.By = CVNS.BPDS.ValueGenerator.Method.StoredProcedure
   StatewicIDValue.Function = "GetNextWicId"
   StatewicIDValue.Parameters.Add("MachineId", "WebService")
   StatewicIDValue.Parameters.Add("Statewicid", "")
   StatewicIDValue.On = CVNS.BPDS.ValueGenerator.Mode.Insert
   StatewicIDValue.OutputPropertyName = "Statewicid"

   With GetMappings
     .TableName = "MEMBER"
     .Add(New FieldMapping("StateWicId", "STATEWICID", True, "String", DataType.VarChar, 8,
StatewicIDValue))
     .Add(New FieldMapping("MedicalHomeId", "MEDICALHOMEID", False, "Int32", DataType.Numeric, 0,
FieldMapping.FieldTypeEnum.ftTranslateZeroToNull))
     .Add(New FieldMapping("HouseholdId", "HOUSEHOLDID", False, "String", DataType.VarChar, 8))
     .Add(New FieldMapping("AgencyId", "AGENCYID", False, "String", DataType.VarChar, 3))
     .Add(New FieldMapping("ClinicId", "SERVICESITEID", False, "String", DataType.VarChar, 3))
     .Add(New FieldMapping("ApplicationDate", "APPLICATIONDATE", False, "Date", DataType.DateTime, 0))
     .Add(New FieldMapping("LastName", "LASTNAME", False, "String", DataType.VarChar, 25))
     .Add(New FieldMapping("FirstName", "FIRSTNAME", False, "String", DataType.VarChar, 20))
    .Add(New FieldMapping("MiddleInitial", "MIDDLEINITIAL", False, "String", DataType.VarChar, 1))
   .
   .
   .
  End With
End Function

End Class
```

The stored procedure in turn retrieves the next available number from the appropriate consignment table and increments the last used value accordingly:

```
CREATE  PROCEDURE [dbo].[GetNextWicId]
    @MachineId varchar(100),
    @StateWicId varchar(8) output
 AS
BEGIN
    DECLARE @LastUsedId int, @StartingId int
```

```
    DECLARE @EndingId int, @NewStartingId int, @NewEndingId int, @MaxEndingId int, @NumOpenBlocks int
    DECLARE @Id int
    SET @Id = -1
    set SET TRANSACTION ISOLATION LEVEL READ COMMITTED

    -- Get the block that contains the current range of reserved ids for the specified machine
    -- We constrain the query by 'LastUsedId < EndingId' to ensure that we do not retrieve a block in
    -- in which the range is "used up".
    -- We issue the subselect to ensure that we only get the current block (i.e. the record having the
lowest range of ids).
    -- since it is possible to have up to 2 records returned from the query. This would happen if the
web service reserved
    -- another block was reserved because the number of available ids in the current block fell below
the minimum
    -- threshold: (EndingId - LastUsedId) < MinThreshold.
    select @StartingId = StartingId, @LastUsedId = LastUsedId, @EndingId = EndingId from
StateWicIdConsignment (updlock) where StartingId = (select min(StartingId) from StateWicIdConsignment
where MachineId = @MachineId and (LastUsedId < EndingId) )

    -- increment the last used
    IF @LastUsedId = 0
        -- Block has not been used yet
        SET @Id = @StartingId
    ELSE
        SET @Id = @LastUsedId + 1
    update StateWicIdConsignment set LastUsedId = @Id,modifydttm=GetDate() where StartingId  =
@StartingId

    -- see we need to add a new block
    IF @MachineId = 'WebService'
    -- we only allocate new blocks on the fly for the web service, blocks for other machines are
allocated while checking out data via a data sync software
    BEGIN
        IF @EndingId - @LastUsedId < 500
        BEGIN
            -- get a count of available blocks for this machine id
            select @NumOpenBlocks = count(*) from StateWicIdConsignment where MachineId =  @MachineId
and (LastUsedId < EndingId)
            IF @NumOpenBlocks =1
            BEGIN
                -- we only have one available block for the web service machine id and we are past
the threshold of minimum availble ids so add a new block
                select @MaxEndingId = max(EndingId) from StateWicIdConsignment
                SET @NewStartingId = @MaxEndingId + 1
                SET @NewEndingId = @NewStartingId + 10000
                insert into StateWicIdConsignment (MachineId, StartingId, LastUsedId, EndingId)
values (@MachineId, @NewStartingId, 0, @NewEndingId)
            END
        END
    END

    -- copy the int into a varchar so the SEt will concat strings insted of implicitly
    -- converting the varchar of zeros to an int and add the values.
    DECLARE @IdAsString varchar(8)
    DECLARE @val1 varchar(8)
    SET @IdAsString = '00000000'
    SET @val1 = @Id
    SET @IdAsString =  REPLACE(SPACE(8-LEN(@Id)),' ','0') + @val1
    Set @StateWicId = @IdAsString
    RETURN 0
END

GO
```

Notice that the stored procedure only allocates a new block "on-the-fly" for the WebService MachineId. This is because this is the only machine that interacts directly with the centralized database which is where all the consignment blocks are maintained. Other machines must create blocks while checking out clinic data in order to ensure that each block of numbers is unique.

While the new member record is being inserted into the database by the Data Access Layer (i.e. the CVNS.BPDS.DataAccess.SqlSeverHelper class) the helper invokes a call to the stroed procedure to retrieve the next id which is then included in the insert statement issued to the database.

There are a number of consignment blocks that are used throughout the system they can easily be identified by looking for tablenames that end with "Consignment" there corresponding stored procedures cab be identified by looking for stored procedures that start with "GetNext".

Mapping classes using these consignment blocks can be identified by scanning the code base for classes ending with "Mapping" that use a ValueGenerator.

12.11 CVNS.BPDS.Security.CryptographicProvider

Provides a simplified interface for encryption/decryption of a specified value as well as the ability to hash a value. The class provides methods to generate a non-weak random key/vector pair which can be used to encrypt and decrypt values. Note the consumer of this class must retain the key/value pair in a secured environment since the same pair is needed to decrypt and encrypted value. The class uses the TripleDESCryptoServiceProvider class from the .NET framework to which implements the Data Encryption Standard (DES) algorithm.

Encrypting and Decrypting Sensitive Information
In order to ensure that no one software component can independently encrypt or decrypt sensitive information the functionality and key information has been distributed across different components of the software.

As mentioned above, a Triple DES algorithm is used to for encryption. The key and vector values used during encryption are stored in the wic.dat file also located in the same folder as the software assemblies. The wic.dat file contains a lengthy string of random characters. Embedded within this string are the key and vector values. The key is scattered within the file in 3 parts while the vector is scattered in 2 parts.

The only software component in the system that can reassemble the key and vector values is the GetParts() method of the Wic.Common.Utilities class. This class takes 2 string parameters by reference and updates the parameters with the key and vector values. The method itself is obfuscated in that there is no mention of any security context within the methods source code.

Software components needing to encrypt or decrypt values must implement code similar to the following:

```
Dim p1 As String
Dim p2 As String
Dim cp As New CVNS.BPDS.Security.CryptographicProvider
Wic.Common.Utilities.GetParts(p1, p2)
.
.
EncryptedValue = cp.Encrypt(Value, p1, p2)
.
.
DecryptedValue = cp.Decrypt(EncryptedValue, p1, p2)
```

For clarity sake the key and vector values being used are included bloew:

Key: `otw9DHk3SVdwtS69TxYsaK/g1HjIf9l/`
IV: `DiycThta3I0=`

12.12 Central Data Store

The Central Data Store is the primary data source and is a single MS SQL Server database used to store all data.
The Central Data Store is a MS SQL Server database used as the interactive data source for all the WIC applications. (Note the some applications can operate in a disconnected mode. In this case the Central Data Store is replaced with the Local Data Store.

12.13 Local Data Store

The Local Data Store is a MS SQL Server database used as an interactive data source for the applications capable of operating in a disconnected mode.

12.14 Service Agents

Service Agents are used by application capable of running in a disconnected mode. These agents route SOA style requests to either the Central Data Store via the Web Services or to the Local Data Store via the local copy of the Business Service Layer software components. The SessionSettings file is used by the service agent components to determine the mode of operation and location of the Web Services.

12.15 Permission Testing

Permissions are expressed as access levels granted to a system features. There are 3 access levels; None, View, and Full Control. Each is mutually exclusive of the other.

Below are 3 examples of how to perform a permission test. In the first 2 examples we are performing an implicit test where as in the second example we are performing an explicit test. Explicit tests are preferred unless logic and/or specifications dictate otherwise.

Example 1

```
If oUser.HasPermissionGreaterThan(pSearchStatewide, alNone) Then
    ' if the current user's permission level for the Statewide search feature
    ' is greater then 'None' then enable the search option
    Me.rdoStatewide.Enabled = True
Else
    Me.rdoStatewide.Enabled = False
End If
```

Example 2

```
If oUser.HasPermission(pSearchStatewide, alNone) Then
    ' if the current user's permission level for the Statewide search feature
    ' is 'None' then disable the search option
    Me.rdoStatewide.Enabled = False
Else
    Me.rdoStatewide.Enabled = True
End If
```

Example 3

```
If oUser.HasPermission(pSearchStatewide, alView) Or _
  oUser.HasPermission(pSearchStatewide, alFullControl) Then
   ' if the current user's permission level for the Statewide search feature
   ' is either 'View' or 'Full Control' then enable the search option
    Me.rdoStatewide.Enabled = True
Else
    Me.rdoStatewide.Enabled = False
End If
```

Permission controls within an interface are stipulated in the conceptual designs (DFDDs) and supporting detailed design documents.

12.16 Implementing Business Rules Checks

Business rules will be implemented as a class exposing a number of methods that encapsulate the records found in the STATEBUSINESSRULES database table. These class methods are provided in order to strongly type the rule evaluation thereby relieving the developer from having to use literal values during the rules checking. The following examples illustrate the usage of the BusinessRules class.

Example 1 – Getting values from the rules

```
Me.lblCounty.Text = oBusinessRules.GetCountyLabel() ' Returns County or Quadrant
```

Example 2 – Testing a Boolean condition

```
If oBusinessRules.ShowTermInfo() Then
    Me.lblTerminationDateText.Visible = True
    Me.lblTerminationDateValue.Visible = True
    Me.lblTermReasonText.Visible = True
    Me.lblTermReasonValue.Visible = True
Else
    Me.lblTerminationDateText.Visible = False
    Me.lblTerminationDateValue.Visible = False
    Me.lblTermReasonText.Visible = False
    Me.lblTermReasonValue.Visible = False
End If
```

Example 3 – Testing a Boolean condition using an enumeration

```
If oBusinessRules.AssignRiskFactor(rf135) Then
    ' add the risk factor if the conditions are met.
End If
```

12.17 Exception Handling

Methods that are capable of throwing exceptions will be wrapped in a `try...catch` construct. All exception caught at the UI layer will be handled by the default handler provided by CVNS.BPDS.Windows.Forms.Form..

```
Try
    _oPD.Verify()
    Dim s As String
    s = _oPSA.SaveDemographics(_oPD)
    If s.Length > 0 Then
        ' server side error occured, diaplay the message
        MsgBox(s)
    End If
Catch ex As Exception
    ' use default exception handler
    Me.HandleException(ex, me.text)
End Try
```

12.18 Summary Lists

Applications have occasion to retrieve and display lists of information. Normally speaking the information being displayed is either a subset or summarization of a significantly larger set of data. Consider a list of names of participants where the first and last name being displayed are only 2 attributes of 20 that are associated with the participant. When retrieving this list we only want to return the information needed to display and uniquely identify the participant. For arguments sake let's say that would be Participant Id, First name, and Last name. There are 2 techniques that can be used to implement a solution for this requirement using our data access framework.

The first technique involves creating a specialized set of classes to represent the list item and database mapping, along with a strongly typed collection to hold these objects. The specialized class will only support the properties ParticipantId, Firstname and Lastname. We would name this class, (derived from BusinessObject), ParticipantListItem so as not to be confused with the fully attributized Paricipant class. The corresponding mapping, (derived from MappingBase), would then be named ParticipantListItemMapping and the collection, (derived from BusinessObjectCollection) would be named ParticipantListItems.

The second technique involves creating a fully attributized Participant class along with its corresponding ParticipantMapping and Participants collection. The developer would provided code that would remove the unwanted FieldMapping items from the Participant objects FieldMappings property at runtime thereby reducing the map to contain only the 3 field maps for ParticipantId, Firstname, and Lastname.

If the list must display data from more than one data source, (i.e. joining 2 or more database tables), as is the case when translating codes to meaningful descriptions (i.e. M to male and F to female), then a database view should be created that provides the aggregated data. In this situation, using the first technique is the preferred choice.

12.19 Building "Persistable" Objects

In the section above we refer to the fully attributized Participant class, also derived from BusinessObject, the FieldMappings for this class are tied to a table rather than a view, this allows the class to be persisted to the data source via a DBHelper object. (For details regarding database access please refer to the Data Acquisition section of this document.)To follow our earlier list items scenario, once a list item is selected the fully attributized object is retrieved using the unique identifier for the item, in this case the participant id. This class provides the coded values or enumerated lists as properties when applicable (i.e. Gender as M or F). Consumers of these objects then translate codes to descriptions by retrieving domain lists and looking up and translating values to descriptions. Loading, Saving, and Deleting these objects from the persistence layer then becomes as simple as passing the object to the appropriate method of a DBHelper object.

12.20 Using Duncan to Generate "Persistable" Objects

A Code generation tool has been developed by the BPDS Architecture team that facilitates the construction of objects that map to database tables and views. The .NET Add-in tool, called Duncan, allows the developer to right-click on a table or view in the .NET IDE Server Explorer and select the Create Table Mapping menu item from the popup menu. Selecting the menu item displays the Mapping window. The developer can then change the mapping information and target class names if needed. Pressing the Ok button dismisses the window and creates the target classes in the selected project in the .NET IDE Solution Explorer.

12.21 Business Object Validation and Broken Rules

Most classes have one or more "edits" that need to be passed before the data can be saved or is considered valid. Classes derived from BusinessObject inherit a Verify() method that can be implemented to include these edits as well as a BrokenRules collection that can be used to record

offensive conditions. For example, assume that Lastname is a required property of a Participant object. The verify method of the class would implement a test to ensure that the length of the Lastname property is greater than 0, if not, the method would add a broken rule the object's BrokenRules collection and then throw an exception of type BusinessObjectException. The consumer of the object would then catch the exception, interrogate the broken rules and take appropriate action. Note that he DBHelper's save method uses reflection to invoke the passed object's Verify() method prior to attempting to save the object to the persistence layer. Consumers can also programmatically invoke the object's Verify() method. This allows the interface layer to invoke the "edits" from the objects in the business service layer. This concept can be applied to workflow classes that may verify relational "edits" across constituent objects.

12.22 Using Database Transactions

A logical transaction should be used when persisting data that spans more one object. In the example below, a transaction is used to group the persistence of a Member and CertContact object. Notice the use of the try…catch in relation to the commit and rollback.

```
Try
    ' Load the atomic objects to be updated
    Member.StateWicId = Demographics.StateWidId
    DBHelper.Load(Member)
    CertContact.StateWicId = Demographics.StateWidId
    DBHelper.Load(CertContact)
    ' update the properties accordingly
    Member.FirstName = Demographics.Person.FirstName
    CertContact.CertStartDate = Demographics.CertStartDate
Catch boe As BusinessObjectException
    Throw boe
Catch ex As Exception
    Throw ex
End Try

' update the member and certcontact info as a single transaction
Try
    With DBHelper
        .BeginTrans()
        .Save(Member)
        .Save(CertContact)
        .CommitTrans()
    End With
Catch boe As BusinessObjectException
    Throw boe
Catch ex As Exception
    DBHelper.RollbackTrans()
Finally
    DBHelper.Dispose()
End Try
```

12.23 Resources

Commonly used strings and images will be placed in a single resource file. Strings should be stored in the resource file with the appropriate tokens embedded in the string: "%1 is a required value." These images and resources are to be assigned a unique id and accessed via an global object that provides an interface for retrieving said items. (see the MSDN article http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptutorials/html/image_resources.asp for details regarding creating a resource file.) used by all the components.

12.24 Messages

Reusable Messages will be stored in a common resource file. Messages will be displayed using the windows msgbox() routine.

12.25 Unit Testing

Unit testing is a critical piece of the software construction process. Complete and thorough unit testing reduces the number of defects encountered during system test. In order to ensure that thorough unit testing will be performed, developers will be required to submit unit test plans prior to constructing the software. These plans must be reviewed and approved before construction begins. Unit testing will be implemented as a series of NUnit test jigs. Please refer to the document entitled *NUnit Test Jig Primer* for a detailed description of how to construct these test jigs.

These unit test jigs will be incorporated into the master build process and will be run after a successful build.

12.26 Coding Guidelines

The Spirit project will use the "VB.Net Guidelines" outlined by the BPDS Architecture team as of Jan 1, 2005. A copy of these guidelines has been copied into the "Development Processes and Procedures" folder the Spirit Repository. In addition, the addenda document "Spirit Coding Guidelines" has been drafted to stipulate the project specific modifications that will be made to the aforementioned guidelines published by the BPDS Architecture team to add additional readability and clarity to the code base.

12.27 Version Control

The following software artifacts will not be versioned in the repository

- The project's bin folder
- The project's obj folder
- The project's .vbproj.user file

When adding a new project to the repository the above items are to be excluded. Note that the _svn folder is used by the version control system and should not be tampered with.

12.28 Code Documentation

12.28.1 Generating External Documentation

External Documentation (Programmer reference material) will be preceded by 3 ticks (''') and commented with XML tags. This documentation will be provided to developer's utilizing the assembly. The xml documentation will be generated using the VB.Doc. The reference documentation will complied using NDoc. Both the XML and the complied reference material

will be built as part of the master build process and will be placed in a common folder within the appropriate project.

The xml tags that can be used can be found in NDoc users guide located at http://ndoc.sourceforge.net/usersguide.html. These tags are based on (and should be the same as) the tags supported in Microsoft's C# language. The following tag usage is in effect for all publicly accessible types and members.

Tag	*Class*	*Property*	*Method*	*Event*
<summary>	R	R	R	R
<remarks>	O	O	O	O
<param>			R*	
<event>			R*	
<returns>			R	
<overloads>		O	O	O
<example>	O	O	O	O
<code>	O	O	O	O
<seealso>	O	O	O	O

R Required
R* Required if applicable
O Optional

<summary>	This tag is the primary description used by IntelliSense and the Object Browser in VisualStudio, and most other development tools
<remarks>	This tag is used to add additional information about a type or member, supplementing the information specified with <summary>.
<param>	Describes a parameter of a method.
<event>	Describes an event raised by a method.
<returns>	This tag describes the return value of a method.
<overloads>	Describes overloaded feature
<example>	Use to provide the descriptive text associated with a usage example
<code>	Used wthin an <example> tag to provide a code snippet.
<seealso>	Used to specify links to other classes or members.

12.28.2 Internal Documentation

Internal documentation will be provided as free form comments preceded by a single tick (‘). These comments should used liberally to explain and/or document the related code for future reference by developer's enhancing or maintaining the code. The purpose of these comments is to provide insight into the reasons, relationships, or rules that are being implemented that may not be apparent by reading the code.

13 Disaster Recovery Provisions

The Disaster Recovery Plan is a SPIRIT and CNI deliverable. The outline of the plan for the data center which will service the SPIRIT system has been attached as part of this DTSD. For security purposes the main body of the Disaster Recovery Plan and COOP have been excluded for general

review as a component of this DTSD. A copy of this plan resides in the Chickasaw Nation Office of the CIO and in the Chickasaw Nation (CNI) Data Center. The detailed elements of this plan which pertain to the SPIRIT implementation can be reviewed by members of the SPIRIT consortium or approved third-parties upon request by contacting the CNI Director of Information Technology at 580-272-5000.

14 Appendix A

The following is a list of tables that contain reference data and are potential candidates to be included in the Reference Data Synchronization process.

Appointments
APPOINTMENTRESOURCE
APPOINTMENTTYPE
BUSINESSDAY
BUSINESSHOUR
CLASSSCHEDULE
CLASSTYPE
DEFAULTDURATION
HOLIDAY
RESOURCESCHEDULE

Food Prescriptions
BASEFOODCATEGORY
BASEFOODCOMBINATIONMAXIMUM
FOODITEM
FOODDISTRIBUTIONITEM
FOODITEMFORMULA
FOODITEMQTY
FOODPACKAGE
FOODPACKAGEBASEFOODCAT
FOODPACKAGEITEM
MILKSUBSTITUTION

Growth Grid Reference Data
BMI
HEAD
HTAGE
INTRAUTERINEWEIGHT
LENGTHWEIGHTAGE
LENGTHWEIGHTTYPE
LGTHAGE
PREMATURELENGTHWEIGHTAGE
PRENATALWEIGHTGAIN
PREPREGWEIGHTHEIGHT
WEIGHTLENGTH
WTAGE
WTHEIGHT
WTLENGTH

General
AGECATAGORY
AGENCY
ANEMIACUTOFF
COUNTY
COUNTYCITYBYZIP

DIAGECATEGORY
DIFOODCATEGORY
DIRECOMMENDATION
FIXEDLOCATION
HEALTHFACILITY
IMMUNIZATIONSERIES
INCOMEELIGIBILITYAMOUNT
INVALIDBFCOMBO
ITEMPURCHASELINKAGE
ITEMPURCHASESIZE
LAZERLABELS
LCLOTHPROGRAM
LEGALMUNICIPALITY
LOCALMUNICIPALITY
LOCALUSECAPTION
LOCALUSECODE
LOCATION
MEDICALHOME
OUTREACHAGENCY
PRIORITYPERWICSTATUS
RACEETHNICITY
REFAGENCYCLINIC
REFAGENCYLOCAL
REFAGENCYSTATE
REFDICTIONARYCATEGORY
REFERENCEDICTIONARY
RISKFACTOR
RISKFACTORREFERENCE
SERVICESITE
SMOKINGAMOUNT
STATEBUSINESSRULES
STATEUSECAPTION
STATEUSECODE
VACCINE
WICSTATUS
WLAGECATEGORY
WLELIGIBILTYCRITERIA
WLELIGIBILITYPROFILE